NEW CONSTRUCTIONS OF CRYPTOGRAPHIC PSEUDORANDOM FUNCTIONS

A Thesis
Presented to
The Academic Faculty

by

Abhishek Banerjee

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in
Algorithms, Combinatorics, and Optimization

School of Computer Science
Georgia Institute of Technology
August 2015

NEW CONSTRUCTIONS OF CRYPTOGRAPHIC PSEUDORANDOM FUNCTIONS

Approved by:

Professor Chris Peikert, Advisor
School of Computer Science
Georgia Institute of Technology

Professor Alexandra Boldyreva
School of Computer Science
Georgia Institute of Technology

Professor Santanu Dey
School of Industrial and Systems Engineering
Georgia Institute of Technology

Professor Lance Fortnow
School of Computer Science
Georgia Institute of Technology

Professor Richard J. Lipton
School of Computer Science
Georgia Institute of Technology

Professor Alon Rosen
School of Computer Science
IDC Herzliya

Date Approved: 29 June 2015

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Pseudorandom functions (PRFs) are the building blocks of symmetric-key cryptography. Almost all central goals of symmetric cryptography (e.g., encryption, authentication, identification) have simple solutions that make efficient use of a PRF. Most existing constructions of these objects are either (a) extremely fast in practice but without provable security guarantees based on hard mathematical problems [AES, Blowfish etc.], or (b) provably secure under assumptions like the hardness of factoring, but extremely inefficient in practice.

Lattice-based constructions enjoy strong security guarantees based on natural mathematical problems, are asymptotically and practically efficient, and have thus far even withstood attacks by quantum algorithms. However, most recent lattice-based constructions are of public-key objects, and it's natural to ask whether these advantages can be brought to the world of symmetric-key constructions.

In this thesis, we construct asymptotically fast and parallel pseudorandom functions basing their security on a well known hard lattice problem called the *learning with errors* problem. We provide several types of constructions that have their respective efficiency and security advantages. In addition to this, we also provide improved constructions of *key-homomorphic PRFs* that achieve almost optimal quasi-linear magnitudes of public parameters, key sizes and incremental run times. We also propose a new cryptographic primitive, *constrained key-homomorphic PRFs*, provide secure candidate constructions and applications. Lastly, we detail an implementation in software of a candidate PRF and analyze its efficiency and security.

CHAPTER I

INTRODUCTION

The focus of this thesis is on *pseudorandom function* (PRF) families, first rigorously defined and constructed by Goldreich, Goldwasser, and Micali ("GGM") [39]. A PRF family is of the form $\mathcal{F} = \{F_s\}$, where the functions $F_s : D \to R$ are all *deterministic* and indexed by the *seed* or the *secret key* s. We typically think of the domain $D = \{0,1\}^k$, and call k the *input length* of the PRF family. Informally, $\mathcal{F}$ is pseudorandom if no efficient adversary, given adaptive oracle access to a randomly chosen function from the family, can distinguish it from a uniformly random function.

1.1 Applications of Pseudorandom Functions

Pseudorandom function families are central objects in symmetric cryptography. Given a PRF family, most central goals of symmetric cryptography (e.g., encryption, authentication, identification) admit simple solutions that make efficient use of the PRF. We detail how two parties sharing a common secret key s would perform these tasks below. (Note that we need the size of the domain to be superpolynomial in the security parameter to prevent birthday attacks in all the applications below.)

- **Encryption:** A message $m \in R$ is encrypted by choosing a random $x \in D$ and sending $(x, m \oplus F_s(x))$ as the ciphertext. The ciphertext (x, y) is consequently decrypted as $x \oplus F_s(x)$. This scheme satisfies indistinguishability against chosen message attack.

- **Authentication:** For a message m that is hashed to some $x \in D$, the authentication tag is simply $F_s(x)$. This scheme is unforgable against chosen message attack.

- **Identification:** To identify themselves to each other on a public channel, the parties run the following protocol: party A sends some randomly chosen $x \in D$ to party B, who responds with $F_s(x)$, which A can then verify. They then run the protocol in the other direction.

See [38, 58] for more details on these applications and others, including authenticated encryption and cryptographic hash functions. Luby and Rackoff [59] show how to construct *pseudorandom permutations* from pseudorandom functions by a construction built using the Feistel cipher. Pseudorandom permutations are the idealized abstractions of block ciphers (which are simply pseudorandom permutations with an efficient inverting algorithm), used for format preserving encryption, among other uses. The cipher block chaining (CBC) mode of operation of a block cipher employs the efficient inversion algorithm in its decryption phase.

PRF families find use in a variety of applications outside of symmetric cryptography. Some of these are detailed below.

- **Public Key Cryptography:** Using pseudorandom functions, Goldreich [37] managed to make the signing algorithm in any stateful signature scheme stateless. Bellare and Goldwasser [14] use PRFs and non-interactive zero knowledge (NIZK) proofs to create digital signatures secure against adaptive chosen message attack.

- **Complexity Theory:** Razborov and Rudich [85] famously proved in an Gödel award-winning paper that if pseudorandom functions exist then a wide class of proof strategies, called *natural proofs*, cannot be used to prove $P \neq NP$.

- **Learning:** In a celebrated result, Valiant [91] proves that the existence of PRFs in any complexity class C implies that there exist concept classes in C that cannot be learned under membership queries (PAC learning).

Variants. In addition to the applications above involving direct use of a PRF, we also mention the following variants that enable a wider variety of applications.

1. *Verifiable Random Functions* (VRFs), first defined by Micali, Rabin and Vadhan [65]: in this setting, there is a public key that implicitly fixes all function values, and the holder of the PRF key s can produce non-interactive proofs of the fact that $y = F_s(x)$ which can be verified using only the public key (along with the PRF input and output and the proof). VRFs have a variety of applications, ranging from resettable zero-knowledge proofs [66] and micropayment schemes [67] to updatable zero-knowledge databases [56], among others.

2. *Key-homomorphic Pseudorandom Functions*, first defined and constructed, in the random oracle model, by Naor, Pinkas and Reingold [74], have the property that $F_s(x) + F_t(x) = F_{s+t}(x)$, that is, they are additively homomorphic over their keys. They show how to distribute the operation of a Key Distribution Center using a key-homomorphic PRF.

3. *Constrained Pseudorandom Functions*, concurrently defined by Boneh and Waters [20], Boyle, Goldwasser and Ivan [21] and Kiayias, Papadopoulos, Triandopoulos and Zacharias [49], are constructions where it is possible to derive constrained keys that enable the evaluation of the PRF on a certain subset of the inputs only. Among a variety of applications of this primitive are identity-based key exchange and broadcast encryption.

1.2 Generic and Algebraic Constructions of Pseudorandom Functions

Generic Constructions of PRFs. The seminal GGM construction is based generically on any length-doubling pseudorandom generator (PRG), which in turn can be constructed from any one-way function [42, 16]. This construction is exceedingly simple to describe: the PRF seed is used as the initial input to the PRG, and each

bit of the PRF input is used to choose one half of the generator output, which is the input to the generator for the next bit of the PRF input. More formally, given a PRG $G\colon \{0,1\}^{\ell} \to \{0,1\}^{2\ell}$, the function family $\mathcal{F} = \{F_s\colon \{0,1\}^{k} \to \{0,1\}^{\ell}\}$ is indexed by $s \in \{0,1\}^{\ell}$ and defined as

$$F_s(x_1 \dots x_k) := G_{x_k}(\dots G_{x_1}(s) \dots),$$

where G_b represents the first or second half of the output, as indicated by the bit b.

We note that the GGM construction above requires k *sequential* invocations of the generator when operating on k-bit inputs. Naor and Reingold [76] manage to construct parallelizable PRFs with depth logarithmic in the input length. They do this by defining a generic object called a "pseudorandom synthesizer" and combining these objects in a recursive binary tree-like fashion (instead of combining pseudorandom generators sequentially like the GGM construction). Essentially, a pseudorandom synthesizer is a deterministic function $S(\cdot,\cdot)$ that accepts two inputs, and for any m polynomial in the security parameter and inputs $x_1, \dots, x_m, y_1, \dots, y_m$, the $m \times m$ matrix of all possible output values $z_{i,j} = S(x_i, y_j)$ is pseudorandom. Using a pseudorandom synthesizer and two independent PRF instances F_0, F_1 on k input bits each, one obtains a PRF on $2k$ bits as follows

$$F(x_1, \dots, x_{2k}) = S\big(F_0(x_1, \dots, x_k), F_1(x_{k+1}, \dots, x_{2k})\big) \ .$$

The base case of a single-bit PRF is defined as $F_{k_0,k_1}(b) = k_b$. Naor and Reingold also provide several candidate constructions of pseudorandom synthesizers. Most of these constructions are algebraic in nature, and were proven secure under number-theoretic assumptions like the decision Diffie-Hellman (DDH) assumption, and the RSA assumption. Their synthesizers are in TC^0, and hence the resulting PRFs lie in TC^1.

Algebraic PRFs, Applications and Limitations. Naor and Reingold ("NR") [77] first provided direct constructions of PRFs that lie in TC^0. These PRFs are algebraic, and are computed by *exponentiating* a group generator by an input-dependent *subset-product* of secret-key values. NR provide two constructions, one secure under the DDH assumption, and the second secure under the hardness of factoring Blum integers. Naor, Reingold and Rosen [78] provide a construction similar in design and security to the second NR construction, but is an improvement in the following sense: while the NR construction can only output one bit at a time, this construction can efficiently output polynomially many bits.

Naor and Reingold show that the operation of the NR construction can be efficiently distributed among various parties that hold shares of the secret key. They also show that interactive zero knowledge proofs for statements like $y = F_s(x)$ and $y \neq F_s(x)$ can be provided, once the party computing the PRF $F_s(\cdot)$ has committed to the seed s. The highly algebraic structure of the NR constructions leads to several more beautiful applications like oblivious PRFs, which are used for private keyword search [32] and adaptive oblivious transfer and secure computation of set-intersection [45].

A series of followup works [54, 19, 30, 43, 74] also construct algebraic PRFs in the same subset-product and exponentiation framework, employing a variety of hardness assumptions for various goals like security, efficiency in time or space, and other applications (VRFs and key-homomorphic PRFs).

The one significant drawback of these algebraic approaches is that achieving low depth for these constructions requires extensive preprocessing and enormous circuits. Moreover, modular exponentiation of large integers, a subroutine in all these constructions, is a costly operation in practice. This means that such number-theoretic constructions remain only a proof of theoretical feasibility rather than practical utility. Moreover, these functions (or at least their underlying hard problems, which

are all no harder than the discrete log problem) can be broken by polynomial-time quantum algorithms [90].

1.3 Lattices for Symmetric Cryptography?

The past few years have seen significant progress in constructing public-key, attribute-based, identity-based, and homomorphic cryptographic schemes using lattices, e.g., [86, 83, 35, 34, 28, 1, 41] and many more. Part of their appeal stems from provable worst-case hardness guarantees (starting with the seminal work of Ajtai [3]), good asymptotic efficiency and parallelism, and apparent resistance to quantum attacks (unlike the classical problems of factoring integers or computing discrete logarithms).

Perhaps surprisingly, there has been comparatively less progress in using lattices for symmetric cryptography. Promisingly, the *learning with errors* [86] problem yields a pseudorandom generator that can be plugged into the GGM construction to construct PRFs. However, this pseudorandom generator involves sampling discrete gaussian noise from the input randomness, which is a costly operation in practice. Moreover, the natural advantage of lattice-based schemes, that they are relatively efficient and highly parallelizable in a practical sense (i.e., they can be evaluated by small, low-depth circuits), are completely lost when plugging them into a generic sequential construction like GGM.

This motivates the search for specialized lattice-based constructions of symmetric objects, that have comparable parallelism and ideally better efficiency than their algebraic counterparts discussed in Section 1.2. While there has been some partial progress in the form of *randomized* weak PRFs [6] and randomized MACs [84, 50], constructing an efficient, parallelizable (deterministic) PRF under lattice assumptions, remained open for some time now.

1.4 Contributions

In this thesis, we make substantial progress in providing new and direct (non-generic) lattice constructions which are highly efficient and provably secure based on hard lattice problems.

Derandomizing LWE and Pseudorandom Synthesizers. The decision *learning with errors* (LWE) problem, first defined and analyzed by Regev [86], and its ring variant (RLWE), introduced by Lyubashevsky, Peikert and Regev [62], forms the cornerstone of most public-key lattice constructions. The decision LWE problem is to distinguish linear equations with small random perturbations from uniform (see Section 2.3 for a more detailed discussion). As discussed in Section 1.3 above, the hardness of this problem leads to a construction of a pseudorandom generator, but sampling the error using input randomness is a cumbersome step in practice. In Chapter 3, we present a simple derandomization technique for LWE: we effectively generate the error terms deterministically by *rounding* down to a smaller modulus. We call this problem the *learning with rounding* (LWR) problem. We show that for appropriate parameters, LWR is at least as hard as LWE. We also analogously define RLWR and give a reduction from RLWE in the ring setting. Using this new hard problem, we subsequently go on to construct pseudorandom generators and pseudorandom synthesizers in TC^0 and obtain our first lattice-based PRFs by plugging them into the GGM construction and the Naor and Reingold [76] synthesizer-to-PRF construction respectively.

Direct Pseudorandom Functions. In Chapter 4, we provide a more parallel direct construction of PRFs in the subset-product framework proposed by Naor and Reingold [77] and Naor, Reingold and Rosen [78]. Instead of performing a subset-product of secret exponents followed by an *exponentiation* of a group generator, we

perform a similar subset-product of secret matrices or polynomial ring elements followed by a *rounding* operation. The ring-LWE based direct construction lies in TC^0, which matches the best prior results. We prove the security of the direct construction directly from the (ring-)LWE assumption in a novel proof technique, which bolsters the LWR security proof to handle adversarial queries. An artifact of this proof technique is that the LWE approximation factor is exponential in the PRF input length k.

Key-Homomorphic Pseudorandom Functions. Boneh *et al.* [18] give a key-homomorphic PRF through a construction that is a variant of the construction in Chapter 4, and hence inherits the large LWE approximation factors of that construction. In Chapter 5, we propose a family of constructions of key homomorphic PRFs which are much more general and compact than the Boneh *et al.* construction, and achieve significant security and efficiency improvements. In fact, the Boneh *et al.* construction is an instantiation of this construction. Unexpectedly, our ring-based constructions enjoy quasi-linear public parameters and secret-key sizes and incremental run times, making them the first provably secure low-depth PRFs to achieve any of these efficiency measures. We summarize these results in Table 1 below.

Constrained Key-Homomorphic Pseudorandom Functions In Chapter 6, we propose a new cryptographic primitive, *constrained key homomorphic pseudorandom functions*. In addition to being able to derive constrained keys which restrict the PRF operation to subsets of the domain, we also require that these constrain keys are additively homomorphic. We provide a procedure that constrains the keys of the construction from Chapter 5 to *prefixes* of PRF inputs, prove that the obtained constrained keys are homomorphic and also prove the security of this scheme under the LWE assumption.

Table 1: Example instantiations of our key-homomorphic PRF (for input length λ and provable 2^λ security against the best known lattice algorithms) as compared with prior lattice-based PRFs. "KH" denotes whether the construction is key homomorphic. Omitting polylogarithmic $\log^{O(1)} \lambda$ factors, "Key" and "Params" are respectively the bit lengths of the secret key and public parameters; "Time/Out" is the best known runtime (in bit operations) per output bit, where $\omega \in [2, 2.373]$ is the exponent of matrix multiplication; and "Out" is the output length in bits. The quantities in brackets refer to the respective ring-based constructions.

Reference	KH?	Depth	Key	Params	Time/Out	Out
Ch. 3 (GGM)	N	λ	λ [λ]	λ^2 [λ]	λ^2 [λ]	λ [λ]
Ch. 3 (synth)	N	$\log_2 \lambda$	λ^3 [λ^2]	0 [0]	$\lambda^{\omega-1}$ [λ]	λ^2 [λ]
Chapter 4	N	1	λ^5 [λ^3]	0 [0]	λ^4 [λ^2]	λ^2 [λ^2]
[18]	Y	1	λ^3 [λ^3]	λ^6 [λ^4]	λ^5 [λ^3]	λ^2 [λ^2]
Chapter 5	Y	$\approx \log_4 \lambda$	λ [λ]	λ^2 [λ]	λ^ω [λ]	λ [λ]

From Theory to Practice. To demonstrate the practical viability of these new constructions, in Chapter 7, we present an implementation in software and analyze the security of an instantiation of the ring-based subset-product construction from Chapter 4. We note that although the theoretical security proofs do not work for the parameters we choose for these instantiations, a preliminary cryptanalysis suggests robust 128-bit security. In addition, this implementation comes to within 4.5 times the throughput of AES-128. A novel contribution is a fast recursive transformation between the cyclic group representation and the usual polynomial representation of the multiplicative units in the cyclotomic ring modulo 2, which might find use in ring-based constructions elsewhere.

1.5 Organization and Credit

This thesis is organized as follows. In Chapter 2, we cover preliminary definitions and supporting theorems that build a foundation for the rest of this thesis. Definitions and supporting theorems specific to a particular construction are placed in a more

appropriate context as they are needed. In Chapter 3, we derandomize (ring-)LWE and present the resulting pseudorandom synthesizer construction. We next present the subset-product based direct PRF construction in Chapter 4. In Chapter 5, we propose improved constructions of key-homomorphic PRFs (in particular, improving on the work of [18]). We propose the new primitive of constrained key-homomorphic PRFs and present candidate constructions and an application in Chapter 6. Finally, we present a fast and secure implementation in software of a lattice based PRF instantiation in Chapter 7. Each chapter begins with an overview of the results and techniques contained therein. Further discussion, open problems and related work are at the end of each chapter, as needed.

The work in this thesis has appeared in various peer-reviewed cryptographic conferences and workshops. More specifically, Chapters 3 and 4 appreared as [12] (joint work with Chris Peikert and Alon Rosen), Chapter 5 appeared as [11] (joint work with Chris Peikert), Chapter 6 appeared as [10] (joint work with Georg Fuchsbauer, Chris Peikert, Kryszstof Pietrzak and Sophie Stevens), and Chapter 7 appeared as [9] (joint work with Hai Brenner, Gaëtan Leurent, Chris Peikert and Alon Rosen).

CHAPTER II

PRELIMINARIES

2.1 Mathematical Background

In this thesis, we will mainly be dealing with matrices and rings of polynomials over the integers $\mathbb{Z}$. For accuracy, we will often talk of quantities (lengths, norms etc.) in the underlying field of reals $\mathbb{R}$, and this will be apparent from the context. Matrices will be denoted by bold uppercase letters (eg: $\mathbf{A}, \mathbf{E}, \mathbf{S}$ etc.), vectors will be denoted by bold lowercase letters (eg: $\mathbf{s}$, $\mathbf{t}$, $\mathbf{v}$ etc.), and scalars will be denoted by lowercase letters (eg: k, ℓ, m, n etc.). Unless otherwise specified, all vectors in this thesis will be row vectors. Unless specified, all additions and multiplications of elements in the quotient group $\mathbb{Z}_q = \mathbb{Z}/qZ$ are modulo the underlying modulus $q \geq 2$.

Rounding. We define the 'rounding' function $\lfloor \cdot \rceil_p \colon \mathbb{Z}_q \to \mathbb{Z}_p$, where $q \geq p \geq 2$ will be apparent from context, as

$$\lfloor x \rceil_p = \lfloor (p/q) \cdot x \rceil \bmod p.$$

We extend $\lfloor \cdot \rceil_p$ component-wise to vectors and matrices over $\mathbb{Z}_q$. Note that we can use any other common rounding method, like the floor $\lfloor \cdot \rfloor$, or ceiling $\lceil \cdot \rceil$ functions, in Equation 2.1.1 above, with only minor changes to our proofs. In implementations, it may be advantageous to use the floor function $\lfloor \cdot \rfloor$ when q and p are both powers of some common base b (e.g., 2). In this setting, computing $\lfloor \cdot \rfloor_p$ is equivalent to dropping the least-significant digit(s) in base b. We will use a different definition of rounding in Chapter 6 ahead for a technical reason that will become apparent later.

Spectral norm. The *spectral norm* $s_1(\mathbf{S})$ of a real matrix $\mathbf{S}$ is defined as $\max_{\mathbf{u}}\|\mathbf{uS}\|_2 = \max_{\mathbf{v}}\|\mathbf{vS}^t\|_2$, taken over all unit vectors $\mathbf{u}, \mathbf{v}$. Equivalently, it is $\max_{\mathbf{x}}\|\mathbf{xS}\|_2/\|\mathbf{x}\|_2$, taken over all $\mathbf{x} \neq \mathbf{0}$. The spectral norm is clearly submultiplicative: $s_1(\mathbf{S} \cdot \mathbf{T}) \leq s_1(\mathbf{S}) \cdot s_1(\mathbf{T})$.

Lemma 2.1.1. If $\mathbf{S}$ is a binary (i.e., 0-1) m-by-m matrix, then $s_1(\mathbf{S}) \leq m$.

Proof. For any vector $\mathbf{x} \in \mathbb{R}^m$, we have

$$\|\mathbf{xS}\|_2^2 = \sum_{i=1}^{m}\Big|\sum_{j=1}^{m} S_{ij}x_j\Big|^2 \leq m\Big(\sum_{j=1}^{m}|x_j|\Big)^2 \leq m^2\sum_{j=1}^{m}|x_j|^2 = m^2\|\mathbf{x}\|_2^2,$$

where the first equality follows from the definition of the Euclidean norm, the first inequality follows from the triangle inequality and the fact that $|S_{ij}| \leq 1$ for all i, j, and the second inequality follows from Cauchy-Schwarz. This proves the claim. □

Cyclotomic Rings. Throughout this thesis, we let R denote the cyclotomic polynomial ring $R = \mathbb{Z}[z]/(z^n + 1)$ for n a power of 2. (Equivalently, R is the ring of integers $\mathbb{Z}[\omega]$ for $\omega = \exp(\pi i/n)$.) For any integer modulus q, define the quotient ring $R_q = R/qR$. An element of R can be represented as a polynomial (in z) of degree less than n having integer coefficients; in other words, the "power basis" $\{1, z, \ldots, z^{n-1}\}$ is a $\mathbb{Z}$-basis for R. Similarly, it is a $\mathbb{Z}_q$-basis for R_q. We extend the rounding operation $\lfloor\cdot\rceil_p$ by applying it coefficient-wise (with respect to the power basis) to the quotient ring R_q (The ring-based constructions in this thesis may be generalized to arbitrary cyclotomic rings using the tools developed in [63].)

2.1.1 Probability Background: Subgaussians and Discrete Gaussians

For a probability distribution X over a domain D, let X^n denote its n-fold product distribution over D^n. The uniform distribution over a finite domain D is denoted by $U(D)$. We use the notation $\mathrm{negl}(\cdot)$ to denote an arbitrary *negligible* function in its input, one that vanishes faster than the inverse of any polynomial. We say that a probability is *overwhelming* (in an underlying parameter λ) if it is $1 - \mathrm{negl}(\lambda)$.

The discrete Gaussian probability distribution over $\mathbb{Z}$ with parameter $r > 0$, denoted $D_{\mathbb{Z},r}$, assigns probability proportional to $\exp(-\pi x^2/r^2)$ to each $x \in \mathbb{Z}$. It is possible to efficiently sample from this distribution (up to $\mathrm{negl}(n)$ statistical distance) via rejection [35].

Subgaussian random variables. In our constructions it is convenient to analyze the behavior of "error" terms using the standard notion of *subgaussian* random variables. (For further details and full proofs, see [94].) A real random variable X (or its distribution) is subgaussian with parameter $r \geq 0$ if for all $t \in \mathbb{R}$, its (scaled) moment-generating function satisfies $\mathbb{E}[\exp(2\pi t X)] \leq \exp(\pi r^2 t^2)$. By a Markov argument, X has Gaussian tails, i.e., for all $t \geq 0$ we have (where we define $0/0 = 0$ in case $t = r = 0$)

$$\Pr[|X| \geq t] \leq 2\exp(-\pi t^2/r^2).$$

(If $\mathbb{E}[X] = 0$, then Gaussian tails also imply subgaussianity.) Any B-bounded centered random variable X (i.e., $\mathbb{E}[X] = 0$ and $|X| \leq B$ always) is subgaussian with parameter $B\sqrt{2\pi}$. In addition, the discrete Gaussian distribution $D_{\mathbb{Z},r}$ over $\mathbb{Z}$ is subgaussian with parameter r ([8] and [70, Lemma 2.8]).

Subgaussian random variables satisfy *Pythagorean additivity*: if X_1, X_2 are independent subgaussians with respective parameters r_1, r_2, then $X_1 + X_2$ is subgaussian with parameter $\sqrt{r_1^2 + r_2^2}$. By induction this extends to the sum of any finite number of independent subgaussians.

A random real *vector* $\mathbf{x}$ is subgaussian with parameter r if for all fixed unit vectors $\mathbf{u}$, the marginal $\langle \mathbf{u}, \mathbf{x} \rangle \in \mathbb{R}$ is subgaussian with parameter r. (In particular, each coordinate of $\mathbf{x}$ is subgaussian with parameter r.) It follows directly from the definition that the concatenation of independent subgaussians with common parameter r is also subgaussian with parameter r. Pythagorean additivity clearly extends

to subgaussian vectors, by linearity. In addition, if $\mathbf{x}$ is subgaussian with parameter r then $\mathbf{xS}$ is subgaussian with parameter $r \cdot s_1(\mathbf{S})$, since $\langle \mathbf{u}, \mathbf{xS} \rangle = \langle \mathbf{uS}^t, \mathbf{x} \rangle$.

We recall a useful result from the non-asymptotic theory of random matrices [94], which bounds the spectral norm of a matrix with independent subgaussian entries.

Lemma 2.1.2. Let $\mathbf{X} \in \mathbb{R}^{n \times m}$ be a random matrix (or vector) whose entries are drawn independently from (not necessarily identical) subgaussian distributions with common parameter r. There exists a universal constant $C > 0$ such that $s_1(\mathbf{X}) \leq r \cdot C(\sqrt{m} + \sqrt{n})$ except with probability at most $2^{-\Omega(m+n)}$.

We next recall a similar result for rings from [62].

Lemma 2.1.3. Let χ be the distribution over the ring R where each coefficient (with respect to the power basis) is chosen independently from $D_{\mathbb{Z},r}$ for some $r > 0$, and let $t = \omega(\sqrt{\log n})$ denote any function that grows asymptotically faster than $\sqrt{\log n}$. Then in the product of $k \geq 1$ independent samples drawn from χ, every coefficient is bounded in magnitude by $(r\sqrt{n} \cdot t)^k / \sqrt{n}$, except with $\exp(-\Omega(t^2)) = \mathrm{negl}(n)$ probability.

2.2 Cryptographic Definitions

The main security parameter through this thesis is λ, and all algorithms (including the adversary) are implicitly given the security parameter λ in unary. All other parameters and design choices are made such that all the algorithms run in $\mathrm{poly}(\lambda)$ time.

Games and Computational Indistinguishablity. We consider adversaries interacting as part of probabilistic experiments called *games*. For an adversary $\mathcal{A}$ and two games H_0, H_1 with which it can interact, $\mathcal{A}$'s *distinguishing advantage* (implicitly, as a function of λ) is $\mathbf{Adv}_{H_0,H_1}(\mathcal{A}) := |\Pr[\mathcal{A} \text{ accepts in } H_0] - \Pr[\mathcal{A} \text{ accepts in } H_1]|$.

Definition 2.2.1 (Computational Indistinguishability). *We say that games H_0 and H_1 are* computationally indistinguishable*, written $H_0 \overset{c}{\approx} H_1$, if* $\mathbf{Adv}_{H_0,H_1}(\mathcal{A}) = \mathrm{negl}(\lambda.)$ *for any probabilistic polynomial-time $\mathcal{A}$.*

By the triangle inequality, $\overset{c}{\approx}$ is a transitive relation over any $\mathrm{poly}(\lambda)$-length sequence of games. If $H_0 \overset{c}{\approx} H_1$ and $\mathcal{S}$ is any probabilistic polynomial-time algorithm, then the outputs of $\mathcal{S}$ playing in games H_0 and H_1 (respectively) are also computationally indistinguishable.

2.2.1 Pseudorandom Functions

Definition 2.2.2 (Pseudorandom functions). *Let A and B be finite sets, and let $\mathcal{F} = \{F_i \colon A \to B\}$ be a function family, endowed with an efficiently sampleable distribution (more precisely, $\mathcal{F}$, A and B are all indexed by the security parameter λ). We say that $\mathcal{F}$ is a* pseudorandom function *(PRF) family if the following two games are computationally indistinguishable:*

1. *Choose a function $F \leftarrow \mathcal{F}$ and give the adversary adaptive oracle access to $F(\cdot)$.*
2. *Choose a uniformly random function $U \colon A \to B$ and give the adversary adaptive oracle access to $U(\cdot)$.*

To efficiently simulate access to a uniformly random function $U \colon A \to B$, one may think of a process in which the adversary's queries are "lazily" answered with independently and randomly chosen elements in B, while keeping track of the answers so that queries made more than once are answered consistently.

2.3 Learning With Errors and Ring Learning with Errors

We recall the learning with errors (LWE) problem due to Regev [86] and its ring analogue (RLWE) by Lyubashevsky, Peikert, and Regev [62]. These problems form the bedrock of security of most lattice based constructions, and almost all the security proofs in this thesis are based on the conjectured hardness of these problems.

2.3.1 Learning with Errors

For positive integer dimension $n = \text{poly}(\lambda)$ and modulus $q \geq 2$, a probability distribution χ over $\mathbb{Z}$, and a vector $\mathbf{s} \in \mathbb{Z}_q^n$, define the LWE distribution $A_{\mathbf{s},\chi}$ to be the distribution over $\mathbb{Z}_q^n \times \mathbb{Z}_q$ obtained by choosing a vector $\mathbf{a} \leftarrow \mathbb{Z}_q^n$ uniformly at random, an error term $e \leftarrow \chi$, and outputting $(\mathbf{a}, b = \langle \mathbf{a}, \mathbf{s} \rangle + e \bmod q)$. We use the following "normal form" of the *decision*-$\mathsf{LWE}_{n,q,\chi}$ problem, which is to distinguish (with advantage non-negligible in λ) between any desired number $m = \text{poly}(\lambda)$ of independent samples $(\mathbf{a}_i, b_i) \leftarrow A_{\mathbf{s},\chi}$ where $\mathbf{s} \leftarrow \chi^n \bmod q$ is chosen from the (folded) error distribution, and the same number of samples from the uniform distribution $U(\mathbb{Z}_q^n \times \mathbb{Z}_q)$. This form of the problem is as hard as the one where $\mathbf{s} \in \mathbb{Z}_q^n$ is chosen uniformly at random [6].

We extend the LWE distribution to $w \geq 1$ secrets, defining $A_{\mathbf{S},\chi}$ for $\mathbf{S} \in \mathbb{Z}_q^{n \times w}$ to be the distribution obtained by choosing $\mathbf{a} \leftarrow \mathbb{Z}_q^n$, an error vector $\mathbf{e} \leftarrow \chi^w$, and outputting $(\mathbf{a}, \mathbf{b} = \mathbf{aS} + \mathbf{e} \bmod q)$. By a standard hybrid argument, distinguishing such samples (for $\mathbf{S} \leftarrow \chi^{n \times w}$) from uniformly random is as hard as decision-$\mathsf{LWE}_{n,q,\chi}$, for any $w = \text{poly}(\lambda)$. It is often convenient to group many (say, m) sample pairs together in matrices. This allows us to express the LWE problem as: distinguish any desired number of pairs $(\mathbf{A}, \mathbf{B} = \mathbf{AS} + \mathbf{E} \bmod q) \in \mathbb{Z}_q^{m \times n} \times \mathbb{Z}_q^{m \times w}$, for the same $\mathbf{S}$, from uniformly random.

For certain moduli q and (discrete) Gaussian error distributions χ, the decision-LWE problem is as hard as the search problem, where the goal is to find $\mathbf{s}$ given samples from $A_{\mathbf{s},\chi}$ (see, e.g., [86, 80, 6, 69], and [70] for the mildest known requirements on q, which include the case where q is a power of 2). In turn, for $\chi = D_{\mathbb{Z},r}$ with $r = \alpha q \geq 2\sqrt{n}$, the search problem is as hard as approximating worst-case lattice problems to within $\tilde{O}(n/\alpha)$ factors; see [86, 80] for precise statements.[1]

[1] It is important to note that the original hardness result of [86] for search-LWE is for a *continuous* Gaussian error distribution, which when rounded naïvely to the nearest integer does not produce a

2.3.2 Ring Learning with Errors

For a modulus q, a probability distribution χ over R, and an element $s \in R_q$, the ring-LWE (RLWE) distribution $A_{s,\chi}$ is the distribution over $R_q \times R_q$ obtained by choosing $a \in R_q$ uniformly at random, an error term $x \leftarrow \chi$, and outputting $(a, b = a \cdot s + x \bmod qR)$. The normal form of the decision-$\mathsf{RLWE}_{R,q,\chi}$ problem is to distinguish (with non-negligible advantage in λ) between any desired number $m = \text{poly}(\lambda)$ of independent samples $(a_i, b_i) \leftarrow A_{s,\chi}$ where $s \leftarrow \chi \bmod q$, and the same number of samples drawn from the uniform distribution $U(R_q \times R_q)$. We will use the error distribution χ over R where each coefficient (with respect to the power basis) is chosen independently from the discrete Gaussian $D_{\mathbb{Z},r}$ for some $r = \alpha q \geq \omega(\sqrt{n \log n})$.

For a prime modulus $q = 1 \bmod 2n$ and the error distribution χ described above, the decision-RLWE problem is as hard as the search problem, via a reduction that runs in time $q \cdot \text{poly}(\lambda)$ [62]. In turn, the search problem is as hard as quantumly approximating worst-case problems on ideal lattices.[2]

2.4 Gadgets and Bit Decomposition

We now recall some useful background about gadgets and bit-decomposition, first used in [70]. For an integer modulus $q \geq 1$, let $\ell = \lceil \log q \rceil$ and define the "gadget" (column) vector

$$\mathbf{g} = (1, 2, 4, \ldots, 2^{\ell-1}) \in \mathbb{Z}_q^\ell,$$

and the (deterministic) "binary decomposition" function $\mathbf{g}^{-1} \colon \mathbb{Z}_q \to \{0,1\}^\ell$ as follows: identifying each $a \in \mathbb{Z}_q$ with its integer residue in $\{0, \ldots, q-1\}$, let $\mathbf{g}^{-1}(a) = (x_0, x_1, \ldots, x_{\ell-1}) \in \{0,1\}^\ell$ where $a = \sum_{i=0}^{\ell-1} x_i 2^i$ is the binary representation of a.

true discrete Gaussian $D_{\mathbb{Z},r}$. Fortunately, a suitable randomized rounding method does so [81].

[2] More accurately, to prove that the search problem is hard for an a priori *unbounded* number of RLWE samples, the worst-case connection from [62] requires the error distribution's parameters to themselves be chosen at random from a certain distribution. Our constructions are easily modified to account for this subtlety, but for simplicity, we ignore this issue and assume hardness for a fixed, public error distribution.

Note that by definition, $\langle \mathbf{g}, \mathbf{g}^{-1}(a) \rangle = a$ for all $a \in \mathbb{Z}_q$, which explains our choice of notation.[3]

Similarly, for vectors and matrices over $\mathbb{Z}_q$ we define the function $\mathbf{G}^{-1} \colon \mathbb{Z}_q^{n \times m} \to \{0,1\}^{n\ell \times m}$ by applying $(\mathbf{g}^{-1})^t$ entry-wise. Notice that for all $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ we have

$$\mathbf{G} \cdot \mathbf{G}^{-1}(\mathbf{A}) = \mathbf{A}, \quad \text{where} \quad \mathbf{G} = \mathbf{I}_n \otimes \mathbf{g} = \operatorname{diag}(\mathbf{g}, \ldots, \mathbf{g}) \in \mathbb{Z}_q^{n \times n\ell}$$

is the block matrix with n copies of $\mathbf{g}$ as diagonal blocks, and zeros elsewhere.

For the ring R_q, define a suitable "gadget" vector $\mathbf{g} \in R_q^\ell$ and deterministic function $\mathbf{g}^{-1} \colon R_q \to R^\ell$, so that $\mathbf{g}^{-1}(a)$ is "short" and $\langle \mathbf{g}, \mathbf{g}^{-1}(a) \rangle = a$ for all $a \in R_q$. (E.g., we may let $\mathbf{g} = (1, 2, 4, \ldots, 2^{\ell-1}) \in R_q^\ell$ and define $\mathbf{g}^{-1}(a)$ so that each of its R-entries has $\{0,1\}$-coefficients with respect to an appropriate "short" $\mathbb{Z}$-basis of R.) Extend $\mathbf{g}^{-1}$ to row vectors over R_q by applying $\mathbf{g}^{-1}$ entry-wise.

[3]These are just particular definitions of $\mathbf{g}, \mathbf{g}^{-1}$ that we fix for convenience. Our constructions and proofs only require that $\mathbf{g}^{-1}$ be deterministic, and that $\mathbf{g}^{-1}(a)$ be a "short" integer vector such that $\langle \mathbf{g}, \mathbf{g}^{-1}(a) \rangle = a$ for all $a \in \mathbb{Z}_q$. Alternatives include using a signed ternary decomposition, or a larger (or mixed-radix) base; the bounds in our security theorems are easily adapted to such choices.

CHAPTER III

LEARNING WITH ROUNDING AND PSEUDORANDOM SYNTHESIZERS

In this chapter, we define the (ring)-learning with rounding problem and describe the construction of a synthesizer-based family of pseudorandom functions that comes almost immediately from this primitive.

3.1 Overview

We recall that in their work introducing *synthesizers* as a foundation for PRFs [76], Naor and Reingold described a synthesizer based on a simple, conjectured hard-to-learn function. At first glance, this route seems very promising for obtaining PRFs from lattices, using LWE as the hard learning problem. However, a crucial point is that Naor and Reingold's synthesizer uses a *deterministic* hard-to-learn function, whereas LWE's hardness depends essentially on adding *random, independent* errors to every output of a mod-q "parity" function. (Indeed, without any error, parity functions are trivially easy to learn.) Probably the main obstacle so far in constructing efficient lattice/LWE-based PRFs has been in finding a way to introduce (sufficiently independent) error terms into each of the exponentially many function outputs, while still keeping the function deterministic and its key size a fixed polynomial. As evidence, consider that recent constructions of weaker primitives such as symmetric authentication protocols [44, 46, 48], randomized weak PRFs [6], and message-authentication codes [84, 50] from noisy-learning problems are all inherently *randomized* functions, where security relies on introducing fresh noise at every invocation. Unfortunately, this is not an option for deterministic primitives like PRFs.

3.1.1 Derandomizing LWE

To resolve the above-described issues, our first main insight is a way of partially "derandomizing" the LWE problem, i.e., generating the *errors* efficiently and deterministically, while preserving hardness. This technique immediately yields a deterministic synthesizer and hence a simple and parallelizable PRF, though with a few subtleties specific to our technique that we elaborate upon below.

Our derandomization technique for LWE is very simple: instead of adding a small random error term to each inner product $\langle \mathbf{a}_i, \mathbf{s} \rangle \in \mathbb{Z}_q$, we just deterministically *round* it to the nearest element of a sufficiently "coarse" public subset of $p \ll q$ well-separated values in $\mathbb{Z}_q$ (e.g., a subgroup). In other words, the "error term" comes solely from deterministically rounding $\langle \mathbf{a}_i, \mathbf{s} \rangle$ to a relatively nearby value. Since there are only p possible rounded outputs in $\mathbb{Z}_q$, it is usually easier to view them as elements of $\mathbb{Z}_p$ and denote the rounded value by $\lfloor \langle \mathbf{a}_i, \mathbf{s} \rangle \rceil_p \in \mathbb{Z}_p$. We call the problem of distinguishing such rounded inner products from uniform samples the *learning with rounding* ($\mathsf{LWR}_{n,q,p}$) problem. Note that the problem can be hard only if $q > p$ (otherwise no error is introduced), that the "absolute" error is roughly q/p, and that the "error rate" relative to q (i.e., the analogue of α in the LWE problem) is on the order of $1/p$.

We show in Theorem 3.2.2 that for appropriate parameters, $\mathsf{LWR}_{n,q,p}$ is at least as hard as $\mathsf{LWE}_{n,q,\chi}$ for $\chi = D_{\mathbb{Z},r}$ for an error rate $\alpha = r/q$ proportional to $1/p$, giving us a worst-case hardness guarantee for LWR. In essence, the reduction relies on the fact that with high probability, we have $\lfloor \langle \mathbf{a}, \mathbf{s} \rangle + e \rceil_p = \lfloor \langle \mathbf{a}, \mathbf{s} \rangle \rceil_p$ when e is small relative to q/p, while $\lfloor U(\mathbb{Z}_q) \rceil_p \approx U(\mathbb{Z}_p)$ where U denotes the uniform distribution. Therefore, given samples $(\mathbf{a}_i, b_i)$ of an unknown type (either LWE or uniform), we can simply round the b_i terms to generate samples of a corresponding type (LWR or uniform, respectively). (The formal proof is somewhat more involved, because it has to deal with the rare event that the error term changes the rounded value.) In the

ring setting, the derandomization technique and hardness proof based on ring-LWE all go through without difficulty as well. While our proof needs both the ratio q/p and the inverse LWE error rate $1/\alpha$ to be slightly super-polynomial in λ, the state of the art in attack algorithms indicates that as long as q/p is an integer (so that $\lfloor U(\mathbb{Z}_q) \rceil_p = U(\mathbb{Z}_p)$) and is at least $\Omega(\sqrt{n})$, LWR may be exponentially hard (even for quantum algorithms) for any $p = \text{poly}(n)$, and superpolynomially hard when $p = 2^{n^\epsilon}$ for any $\epsilon < 1$.

We point out that in LWE-based cryptosystems, rounding to a fixed, coarse subset is a common method of removing noise and recovering the plaintext when decrypting a "noisy" ciphertext; here we instead use it to avoid having to introduce any random noise in the first place. We believe that this technique should be useful in many other settings, especially in symmetric cryptography. For example, the LWR problem immediately yields a simple and practical pseudorandom generator, as we see ahead in Section 3.3, that does not require extracting biased (e.g., Gaussian) random values from its input seed, unlike the standard pseudorandom generators based on the LWE or LPN (learning parity with noise) problems.

3.1.2 LWR-Based Synthesizers and PRFs

Recall from [76] that a pseudorandom *synthesizer* is a two-argument function $S(\cdot, \cdot)$ such that, for random and independent sequences $x_1, \ldots, x_m$ and $y_1, \ldots, y_m$ of inputs (for any $m = \text{poly}(\lambda)$), the matrix of all m^2 values $z_{i,j} = S(x_i, y_j)$ is pseudorandom (i.e., computationally indistinguishable from uniform). A synthesizer can be seen as an (almost) length-*squaring* pseudorandom generator with good locality properties, in that it maps $2m$ random "seed" elements (the x_i and y_j) to m^2 pseudorandom elements, and any component of its output depends on only two components of the input seed.

Using synthesizers in a recursive tree-like construction, Naor and Reingold gave

PRFs on k-bit inputs, which can be computed using a total of about k synthesizer evaluations, arranged nicely in only $\lg k$ levels (depth). Essentially, the main idea is that given a synthesizer $S(\cdot,\cdot)$ and two independent PRF instances F_0 and F_1 on t input bits each, one gets a PRF on $2t$ input bits, defined as

$$F(x_1 \cdots x_{2t}) = S\big(\, F_0(x_1 \cdots x_t)\,,\, F_1(x_{t+1} \cdots x_{2t})\,\big). \tag{3.1.2}$$

The base case of a 1-bit PRF can trivially be implemented by returning one of two random strings in the function's secret key. Using particular NC^1 synthesizers based on a variety of both concrete and general assumptions, Naor and Reingold therefore obtain k-bit PRFs in NC^2, i.e., having circuit depth $O(\log^2 k)$.

We give a very simple and computationally efficient $\mathsf{LWR}_{n,q,p}$-based synthesizer $S_{n,q,p}\colon \mathbb{Z}_q^n \times \mathbb{Z}_q^n \to \mathbb{Z}_p$, defined as

$$S_{n,q,p}(\mathbf{a}, \mathbf{s}) = \lfloor \langle \mathbf{a}, \mathbf{s} \rangle \rceil_p.$$

Pseudorandomness of this synthesizer under LWR follows by a standard hybrid argument, using the fact that the $\mathbf{a}_i$ vectors given in the LWR problem are public. (In fact, the synthesizer outputs $S(\mathbf{a}_i, \mathbf{s}_j)$ are pseudorandom even given the $\mathbf{a}_i$.) Or for better space and time complexity, we can instead use the ring-LWR synthesizer $S_{R,q,p}(s_1, s_2) = \lfloor s_1 \cdot s_2 \rceil_p$, since the ring product $s_1 \cdot s_2 \in R_q$ is the same size as $s_1, s_2 \in R_q$.

To obtain a PRF using the tree construction of [76], we need the synthesizer output length to exactly match its input length. To this end, we employ an efficient *bijection*, for $m \geq n$ such that $p^m = q^n$, $K\colon \mathbb{Z}_p^{m \times m} \to \mathbb{Z}_q^{n \times m}$. We then modify the definition of the pseudorandom synthesizer function as follows

$$T_{n,p,q}(\mathbf{S}_1, \mathbf{S}_2) := K\Big(\lfloor \mathbf{S}_1^t \cdot \mathbf{S}_2 \rceil_p\Big) \in \mathbb{Z}_q^{n \times m},$$

for $\mathbf{S}_i \in \mathbb{Z}_q^{n \times m}$. Analogously, for $q = p^m$, we use the efficient bijection $K\colon R_p^{m \times m} \to R_q^m$ to redefine the ring-based synthesizer $T_{R,q,p}(\mathbf{s}_1, \mathbf{s}_2) = K(\lfloor \mathbf{s}_1^t \cdot \mathbf{s}_2 \rceil_p)$, for $\mathbf{s}_i \in R_q^m$. These

synthesizers can be plugged into Equation (3.1.2) to give PRFs whose security under (ring-)$\mathsf{LWR}_{n,q,p}$ follows directly from the security of the synthesizers under (ring-)LWR and the security proof from [76].

Note that the matrix multiplication can be done with a constant-depth, size-$O(n^2)$ arithmetic circuit over $\mathbb{Z}_q$. The ring product can also be computed with a constant depth, size-$O(n^2)$ circuit over $\mathbb{Z}_q$, or in $O(\log n)$ depth and only $O(n \log n)$ scalar operations using Fast Fourier Transform-like techniques [61, 62]. Moreover, we see in Section 3.4.4 that these synthesizers are as efficient in parallel depth as those from [76].

3.2 Learning With Rounding

We now define the "learning with rounding" (LWR) problem and its ring analogue, which are like "derandomized" versions of the usual (ring)-LWE problems, in that the error terms are chosen deterministically.

Definition 3.2.1. *Let $\lambda \geq 1$ be the main security parameter, dimension $n = \mathrm{poly}(\lambda)$ and moduli $q \geq p \geq 2$ all be integers.*

- *For a vector $\mathbf{s} \in \mathbb{Z}_q^n$, define the LWR distribution $L_{\mathbf{s}}$ to be the distribution over $\mathbb{Z}_q^n \times \mathbb{Z}_p$ obtained by choosing a vector $\mathbf{a} \leftarrow \mathbb{Z}_q^n$ uniformly at random, and outputting $(\mathbf{a}, b = \lfloor \langle \mathbf{a}, \mathbf{s} \rangle \rceil_p)$.*

- *For $s \in R_q$ (defined in Section 2.3), define the ring-LWR (RLWR) distribution L_s to be the distribution over $R_q \times R_p$ obtained by choosing $a \leftarrow R_q$ uniformly at random and outputting $(a, b = \lfloor a \cdot s \rceil_p)$.*

For a given distribution over $\mathbf{s} \in \mathbb{Z}_q^n$ (e.g., the uniform distribution), the decision-$\mathsf{LWR}_{n,q,p}$ problem is to distinguish (with advantage non-negligible in λ) between any desired number of independent samples $(\mathbf{a}_i, b_i) \leftarrow L_{\mathbf{s}}$, and the same number of samples

drawn uniformly and independently from $\mathbb{Z}_q^n \times \mathbb{Z}_p$. *The decision-*$\mathsf{RLWR}_{R,q,p}$ *problem is defined analogously.*

Note that we have defined LWR exclusively as a decision problem, as this is the only form of the problem we will need. By a simple (and by now standard) hybrid argument, the (ring-)LWR problem is no easier, up to a $\mathrm{poly}(\lambda)$ factor in advantage, if we reuse each public $\mathbf{a}_i$ across several independent secrets. That is, distinguishing samples $(\mathbf{a}_i, \lfloor \langle \mathbf{a}_i, \mathbf{s}_1 \rangle \rceil_p, \ldots, \lfloor \langle \mathbf{a}_i, \mathbf{s}_\ell \rangle \rceil_p) \in \mathbb{Z}_q^n \times \mathbb{Z}_p^\ell$ from uniform, where each $\mathbf{s}_j \in \mathbb{Z}_q^n$ is chosen independently for any $\ell = \mathrm{poly}(\lambda)$, is at least as hard as decision-LWR for a single secret $\mathbf{s}$. An analogous statement also holds for ring-LWR.

3.2.1 Reduction from LWE

We now show that for appropriate parameters, decision-LWR is at least as hard as decision-LWE. We say that a probability distribution χ over $\mathbb{R}$ (more precisely, a family of distributions χ_n indexed by the security parameter λ) is *B-bounded* (where $B = B(\lambda)$ is a function of λ) if $\Pr_{x \leftarrow \chi}[|x| > B] \leq \mathrm{negl}(\lambda)$. Similarly, a distribution over the ring R is B-bounded if the marginal distribution of every coefficient (with respect to the power basis) of an $x \leftarrow \chi$ is B-bounded.

Theorem 3.2.2. *Let χ be any efficiently sampleable B-bounded distribution over $\mathbb{Z}$, and let $q \geq p \cdot B \cdot \lambda^{\omega(1)}$. Then for any distribution over the secret $\mathbf{s} \in \mathbb{Z}_q^n$, solving decision-*$\mathsf{LWR}_{n,q,p}$ *is at least as hard as solving decision-*$\mathsf{LWE}_{n,q,\chi}$ *for the same distribution over $\mathbf{s}$. The same holds true for* $\mathsf{RLWR}_{R,q,p}$ *and* $\mathsf{RLWE}_{R,q,\chi}$*, for any B-bounded χ over R.*

We note that although our proof uses a super-polynomial $q = \lambda^{\omega(1)}$, as long as $q/p \geq \sqrt{n}$ is an integer, the LWR problem appears to be exponentially hard (in n) for any $p = \mathrm{poly}(\lambda)$, and super-polynomially hard for $p \leq 2^{n^\epsilon}$ for any $\epsilon < 1$, given the state of the art in noisy learning algorithms [15, 7] and lattice reduction

algorithms [51, 89]. We also note that in our proof, we do not require the error terms drawn from χ in the LWE samples to be independent; we just need them all to have magnitude bounded by B with overwhelming probability.

Proof of Theorem 3.2.2. We give a detailed proof for the LWR case; the one for RLWR proceeds essentially identically. The main idea behind the reduction is simple: given pairs $(\mathbf{a}_i, b_i) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ which are distributed either according to an LWE distribution $A_{\mathbf{s},\chi}$ or are uniformly random, we translate them into the pairs $(\mathbf{a}_i, \lfloor b_i \rceil_p) \in \mathbb{Z}_q^n \times \mathbb{Z}_p$, which we show will be distributed according to the LWR distribution $L_{\mathbf{s}}$ (with overwhelming probability) or uniformly random, respectively. Proving this formally takes some care, however. We proceed via a sequence of games.

Game H_0. This is the real attack game against the LWR distribution. That is, we choose $\mathbf{s}$ and upon request generate and give the attacker independent samples from $L_{\mathbf{s}}$.

Game H_1. Here the attack is against a 'rounded' version of the LWE distribution $A_{\mathbf{s},\chi}$. That is, we first choose $\mathbf{s}$. Then each time the attacker requests a sample, we generate a pair $(\mathbf{a}, b)$ distributed according to $A_{\mathbf{s},\chi}$ (that is, choose $\mathbf{a} \leftarrow \mathbb{Z}_q^n$ and $b = \langle \mathbf{a}, \mathbf{s} \rangle + x$ for $x \leftarrow \chi$), and return the pair $(\mathbf{a}, \lfloor b \rceil_p)$, but with one exception: we define a 'bad event' Bad to be

$$\mathsf{Bad} := \lfloor b + [-B, B] \rceil_p \neq \left\{ \lfloor b \rceil_p \right\}.$$

That is, Bad indicates whether b is "too close" to some value in $\mathbb{Z}_q$ having a different rounded value. (In other words, rounding the sample $(\mathbf{a}, b)$ from $A_{\mathbf{s},\chi}$ may give a different result than the corresponding sample $(\mathbf{a}, \lfloor \langle \mathbf{a}, \mathbf{s} \rangle \rceil_p)$ from the $L_{\mathbf{s}}$ distribution.) If Bad occurs on any of the attacker's requests for a sample, we immediately abort the game.

If Bad does not occur for a particular sample $(\mathbf{a}, b)$, then we have $\lfloor b \rceil_p := \lfloor \langle \mathbf{a}, \mathbf{s} \rangle + x \rceil_p = \lfloor \langle \mathbf{a}, \mathbf{s} \rangle \rceil_p$ with overwhelming probability over the choice of $x \leftarrow \chi$, because χ is B-bounded. It immediately follows that for any (potentially unbounded) attacker $\mathcal{A}$,

$$\mathbf{Adv}_{H_0,H_1}(\mathcal{A}) \leq \Pr[\mathsf{Bad} \text{ occurs in } H_1 \text{ with attacker } \mathcal{A}] + \mathrm{negl}(\lambda). \tag{3.2.2}$$

We do not directly bound the probability of Bad occurring in H_1, instead deferring it to the analysis of the next game, where we can show that it is indeed negligible.

Game H_2. Here whenever the attacker requests a sample, we choose $(\mathbf{a}, b) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ uniformly at random and give $(\mathbf{a}, \lfloor b \rceil_p)$ to the attacker, subject to the same "bad event" and abort condition as described in the game H_1 above. Under the decision-LWE assumption and by the fact that Bad can be tested efficiently given b, a straightforward reduction implies that $\mathbf{Adv}_{H_1,H_2}(\mathcal{A}) \leq \mathrm{negl}(\lambda)$ for any efficient attacker $\mathcal{A}$. For the same reason, it also follows that

$$|\Pr[\mathsf{Bad} \text{ occurs in } H_1] - \Pr[\mathsf{Bad} \text{ occurs in } H_2]| \leq \mathrm{negl}(\lambda).$$

Now for each uniform b, $\Pr[\mathsf{Bad} \text{ occurs on } b \text{ in } H_2] \leq (2B+1) \cdot p/q = \mathrm{negl}(\lambda)$, by assumption on q. It follows by a union bound over all the samples, and Equation (3.2.2), that

$$\Pr[\mathsf{Bad} \text{ occurs in } H_1 \text{ with } \mathcal{A}] \leq \mathrm{negl}(\lambda) \quad \Rightarrow \quad \mathbf{Adv}_{H_0,H_1}(\mathcal{A}) = \mathrm{negl}(\lambda).$$

Game H_3. This game is similar to the game H_2, with pairs $(\mathbf{a}, b) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ being chosen uniformly at random and Bad being defined similarly. However, in this game we always return $(\mathbf{a}, \lfloor b \rceil_p)$ to the attacker, even when Bad occurs. By the analysis above, we have that for any (potentially unbounded) attacker $\mathcal{A}$,

$$\mathbf{Adv}_{H_2,H_3}(\mathcal{A}) \leq \Pr[\mathsf{Bad} \text{ occurs in } H_3 \text{ with } \mathcal{A}] = \Pr[\mathsf{Bad} \text{ occurs in } H_2 \text{ with } \mathcal{A}] = \mathrm{negl}(\lambda).$$

Game H_4. In this game we give the attacker samples drawn uniformly from $\mathbb{Z}_q^n \times \mathbb{Z}_p$. The statistical distance between $U(\mathbb{Z}_q^n \times \mathbb{Z}_p)$ and $U(\mathbb{Z}_q^n) \times \lfloor U(\mathbb{Z}_q) \rceil_p$ is at most $p/q = \mathrm{negl}(\lambda)$ by assumption on q, so by a union bound over all the $\mathrm{poly}(\lambda)$ samples, we have $\mathbf{Adv}_{H_3,H_4}(\mathcal{A}) = \mathrm{negl}(\lambda)$ for any efficient attacker $\mathcal{A}$.

Finally, by the triangle inequality, we have $\mathbf{Adv}_{H_0,H_4}(\mathcal{A}) = \mathrm{negl}(\lambda)$ for any efficient adversary $\mathcal{A}$, which completes the proof. Essentially the same proof works for the RLWR problem as well. □

3.3 PRFs via the GGM Construction

If parallel complexity is not a concern, and one wishes to minimize the total amount of work per PRF evaluation (or the seed length), then the original GGM construction with an LWR-based pseudorandom generator may turn out to be even more efficient in practice. Recall that the GGM construction makes generic use of any length-doubling pseudorandom generator $G : \{0,1\}^n \to \{0,1\}^{2n}$. The generator's output $G(s)$ is viewed as a pair $(G_0(s), G_1(s))$, where $|G_0(s)| = |G_1(s)| = n$. The key for a member of the PRF family is a seed s for G, and on input $x \in \{0,1\}^k$ the function is defined as

$$F_s(x_1 \cdots x_k) = G_{x_k}(G_{x_{k-1}}(\cdots G_{x_1}(s) \cdots)).$$

As we see ahead, the LWR problem immediately yields a simple and practical pseudorandom generator that, in contrast to the generators obtained from the LWE or LPN problems, does not require extracting biased random error terms from its input seed. By plugging this generator into the GGM construction we immediately get a PRF whose evaluation involves precisely k sequential evaluations of the underlying generator.

The LWR-based generator that we have in mind is a function $G_{\mathbf{A}} : \mathbb{Z}_q^n \to \mathbb{Z}_p^m$, where the moduli $q \gg p$ and the (uniformly random) matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ are publicly

known. Given a seed $\mathbf{s} \in \mathbb{Z}_q^n$, the generator is defined as

$$G_{\mathbf{A}}(\mathbf{s}) = \lfloor \mathbf{s} \cdot \mathbf{A} \rceil_p.$$

The generator's seed length (in bits) is $n \log_2 q$ and its output length is $m \log_2 p$, which gives an expansion rate of $(m \log_2 p)/(n \log_2 q) = (m/n) \log_q p$. For example, to obtain a length-doubling generator we may set $q = p^2 = 2^{2k} > n$ and $m = 4n$. (Other choices yielding different expansion rates are of course possible.) This choice of parameters has the additional benefit of admitting a practical implementation of the rounding and inner-product operations. Note also that when evaluating the resulting PRF, one can get the required part of $G_{\mathbf{A}}(\mathbf{s})$ by computing only the inner products of $\mathbf{s}$ with the corresponding columns of $\mathbf{A}$, not the entire product $\mathbf{s} \cdot \mathbf{A}$.

For an even faster implementation one may replace $G_{\mathbf{A}}$ by its analogous ring variant, obtained by replacing $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ with uniform $\mathbf{a} \in R_q^m$, and $\mathbf{s} \in \mathbb{Z}_q^n$ with uniform $s \in R_q$. We note that the ring variant is particularly efficient to evaluate using Fast Fourier Transform-like algorithms [61, 62]. These efficiency improvements are expounded in much more detail in the case of pseudorandom synthesizers, which we detail in Section 3.4.3 ahead.

3.4 Synthesizer-Based PRFs

We now describe the LWR-based synthesizer and our construction of a PRF from it. We first define a *pseudorandom synthesizer*, slightly modified from the definition proposed by Naor and Reingold [76]; our definition differs from theirs only in that we allow the domain and range to be different.

Let $S : A \times A \to B$ be a function (where A and B are finite domains, which along with S are implicitly indexed by the security parameter n) and let $X = (x_1, \ldots, x_k) \in A^k$ and $Y = (y_1, \ldots, y_\ell) \in A^\ell$ be two sequences of inputs. Then $\mathbf{C}_S(X, Y) \in B^{k \times \ell}$ is defined to be the matrix with $S(x_i, y_j)$ as its (i, j)th entry. (Here $\mathbf{C}$ stands for

combinations.)

Definition 3.4.1 (Pseudorandom Synthesizer). *We say that a function $S : A \times A \to B$ is a* pseudorandom synthesizer *if it is polynomial-time computable, and if for every* poly(λ)*-bounded $k = k(\lambda)$, $\ell = \ell(\lambda)$,*

$$\mathbf{C}_S\big(\, U(A^k)\,,\, U(A^\ell)\,\big) \overset{c}{\approx} U\big(B^{k\times\ell}\big).$$

That is, the matrix $\mathbf{C}_S(X, Y)$ for uniform and independent $X \leftarrow A^k$, $Y \leftarrow A^\ell$ is computationally indistinguishable from a uniformly random k-by-ℓ matrix over B.

3.4.1 Synthesizer Constructions

We now describe synthesizers whose security is based on the (ring-)LWR problem.

Construction 3.4.2 ((Ring-)LWR Synthesizer). For moduli $q \geq p \geq 2$, the LWR synthesizer is the function $S_{n,q,p} \colon \mathbb{Z}_q^n \times \mathbb{Z}_q^n \to \mathbb{Z}_p$ defined as

$$S_{n,q,p}(\mathbf{x}, \mathbf{y}) = \lfloor \langle \mathbf{x}, \mathbf{y} \rangle \rceil_p.$$

The RLWR synthesizer is the function $S_{R,q,p} \colon R_q \times R_q \to R_p$ defined as

$$S_{R,q,p}(x, y) = \lfloor x \cdot y \rceil_p.$$

Theorem 3.4.3. *Assuming the hardness of decision-$\mathsf{LWR}_{n,q,p}$ (respectively, decision-$\mathsf{RLWR}_{R,q,p}$) for a uniformly random secret, the functions $S_{n,q,p}$ (respectively, $S_{R,q,p}$) given in Construction 3.4.2 above are pseudorandom synthesizers.*

Proof. Let $\ell, k = \text{poly}(\lambda)$ be arbitrary. Let $X = (\mathbf{x}_1, \ldots, \mathbf{x}_k)$ and $Y = (\mathbf{y}_1, \ldots, \mathbf{y}_\ell)$ be uniformly random and independent sequences of $\mathbb{Z}_q^n$-vectors. Assuming the hardness of "multiple secrets" version of decision-$\mathsf{LWR}_{n,q,p}$ (see the remark following Definition 3.2.1), we have that the tuples

$$\big(\mathbf{x}_i, \lfloor \langle \mathbf{x}_i, \mathbf{y}_1 \rangle \rceil_p, \ldots, \lfloor \langle \mathbf{x}_i, \mathbf{y}_\ell \rangle \rceil_p\big) \in \mathbb{Z}_q^n \times \mathbb{Z}_p^\ell$$

for $i = 1, \ldots, k$ are computationally indistinguishable from uniform and independent. That is,

$$\big((\mathbf{x}_i)_{i\in[k]}, \mathbf{C}_S(X,Y)\big) \overset{c}{\approx} U(\mathbb{Z}_q^{n\times k} \times \mathbb{Z}_p^{k\times \ell}).$$

From this stronger fact, we have that $\mathbf{C}_S(X,Y) \overset{c}{\approx} U(\mathbb{Z}_p^{k\times\ell})$, as desired. □

Note that the input domain $\mathbb{Z}_q^n$ of $S_{n,q,p}$ is much larger than its range $\mathbb{Z}_p$. To construct PRFs, we will need to *compose* synthesizers, so we need to enlarge the range substantially. As described in [76], this is easily done by extending the domain to a Cartesian product of the original one. Specifically, we extend $S_{n,q,p}$ to map from $\mathbb{Z}_q^{n\times k} \times \mathbb{Z}_q^{n\times\ell}$ to $\mathbb{Z}_p^{k\times\ell}$, for any $k, \ell = \mathrm{poly}(\lambda)$, as follows:

$$S_{n,q,p}(\mathbf{X},\mathbf{Y}) := \mathbf{C}_{S_{n,q,p}}(\mathbf{X}^t, \mathbf{Y}^t) = \lfloor \mathbf{X}\cdot\mathbf{Y}^t \rceil_p. \tag{3.4.4}$$

It follows directly from the definition that the above function is a pseudorandom synthesizer, if the original one is.

Enlarging the range allows us to compose the synthesizer with itself in two ways, which we now describe. The first way involves making the domain and range match *exactly* (which corresponds to the original synthesizer definition of [76]). Let parameters m, n, p, q be such that $p^m = q^n$, and let $K\colon \mathbb{Z}_p^m \to \mathbb{Z}_q^n$ be some efficiently computable bijection, which we extend row-wise to map $\mathbb{Z}_p^{k\times m}$ to $\mathbb{Z}_q^{k\times n}$ for any polynomially sized k.[1] We then define the function $T_{n,q,p}\colon \mathbb{Z}_q^{m\times n} \times \mathbb{Z}_q^{m\times n} \to \mathbb{Z}_q^{m\times n}$ as $T_{n,q,p} = K \circ S_{n,q,p}$. More explicitly,

$$T_{n,q,p}(\mathbf{X},\mathbf{Y}) := K(S_{n,q,p}(\mathbf{X},\mathbf{Y})) = K(\lfloor \mathbf{X}\cdot\mathbf{Y}^t \rceil_p).$$

Because K is an (efficiently computable) bijection, $T_{n,q,p}$ is a pseudorandom synthesizer if $S_{n,q,p}$ is.

[1] For example, if n divides m, i.e., $q = p^{m/n}$, then we have the following "change of base" bijection: for an input vector $\mathbf{x} \in \mathbb{Z}_p^m$, we divide it into n groups of size m/n each, and interpret each group as the base-p representation of a value in $\mathbb{Z}_q$, to obtain a vector $\mathbf{y} \in \mathbb{Z}_q^n$.

In the ring setting, we similarly extend $S_{R,q,p}$ to map $R_q^k \times R_q^\ell$ to $R_p^{k\times\ell}$, and for $q = p^m$ and an efficiently computable bijection $K\colon R_p^m \to R_q$, we define the corresponding domain-preserving synthesizer $T_{R,q,p}\colon R_q^m \times R_q^m \to R_q^m$ as

$$T_{R,q,p}(\mathbf{x}, \mathbf{y}) := K(S_{R,q,p}(\mathbf{x}, \mathbf{y})) = K(\lfloor \mathbf{x} \cdot \mathbf{y}^t \rceil_p).$$

A second way of making the synthesizer $S_{n,q,p}$ composable is to set $k = \ell = n$, yielding a function where the *dimensions* of the domain $\mathbb{Z}_q^{n\times n}$ and range $\mathbb{Z}_p^{n\times n}$ match exactly, but the *moduli* q, p are typically different. By using a tower of decreasing moduli, the corresponding synthesizers can therefore be composed in sequence to yield a PRF. We give further details at the end of the next subsection, but note that this method has substantially weaker security guarantees than the domain-preserving approach.

3.4.2 PRF Constructions

Since the function $T_{n,q,p}$ (respectively, $T_{R,q,p}$) is a domain-preserving synthesizer under the decision-LWR (respectively, decision-RLWR) assumption, we can plug it into the Naor-Reingold construction to get PRFs. For self-containment, we recall the construction here, specialized to our synthesizers. The security of the construction as a PRF family follows from the security of the underlying synthesizer, by [76, Theorem 5.1].

Construction 3.4.4 ((Ring-)LWR PRF). For positive integers $m \geq n$, $q \geq p \geq 2$ such that $q = p^{m/n}$, the LWR family $\mathcal{F}^{(j)}$ for $j \geq 0$ is defined inductively to consist of functions from $\{0,1\}^{2^j}$ to $\mathbb{Z}_q^{n\times m}$.

- For $j = 0$, a function $F \in \mathcal{F}^{(0)}$ is indexed by $\mathbf{S}_b \in \mathbb{Z}_q^{n\times m}$ for $b \in \{0,1\}$, and is defined simply as $F_{\{\mathbf{S}_b\}}(x) = \mathbf{S}_x$. We endow $\mathcal{F}^{(0)}$ with the distribution where the $\mathbf{S}_b$ are uniform and independent.

- For $j \geq 1$, a function $F \in \mathcal{F}^{(j)}$ is indexed by some $F_0, F_1 \in \mathcal{F}^{(j-1)}$, and is defined as

$$F_{F_0,F_1}(x_0, x_1) = T_{n,q,p}\big(\, F_0(x_0)\,,\, F_1(x_1)\,\big)$$

  where $|x_0| = |x_1| = 2^{j-1}$. We endow $\mathcal{F}^{(j)}$ with the distribution where F_0 and F_1 are chosen independently from $\mathcal{F}^{(j-1)}$.

More explicitly, an $F \in \mathcal{F}^{(j)}$ is indexed by a set of uniformly random matrices $\{\mathbf{S}_{i,b}\}$ for $i \in [2^j]$, $b \in \{0,1\}$.

The ring-LWR family $\mathcal{RF}^{(j)}$ is defined similarly, but with $q = p^m$, to consist of functions from $\{0,1\}^{2^j}$ to R_q^m, where in the base case $j = 0$ we replace each $\mathbf{S}_b$ with a uniformly random $\mathbf{s}_b \in R_q^m$, and in the inductive case $j \geq 1$ we use the ring-LWR synthesizer $T_{R,q,p}$. Therefore, a function $RF \in \mathcal{RF}^{(j)}$ is indexed by a set of uniformly random vectors $\{\mathbf{s}_{i,b}\}$ for $i \in [2^j], b \in \{0,1\}$.

In the preliminary version of this work [12], we gave an alternative PRF construction that uses just the synthesizer from Equation (3.4.4) instantiated with $k = \ell = n$, i.e., $S_{n,q,p} \colon \mathbb{Z}_q^{n \times n} \to \mathbb{Z}_p^{n \times n}$. Since the input and output moduli are different, Construction 3.4.4 changes as follows: for the family $\mathcal{F}^{(d)}$, we have a tower of moduli $q_0 \geq q_1 \geq \ldots \geq q_d$. For $j = 0$ the function index is a uniformly random matrix modulo q_0, and for $j \geq 1$ we use the synthesizer S_{n,q_{j-1},q_j}. In the security proof, which proceeds essentially identically to the one from [76], we rely on the conjectured hardness of $\mathsf{LWR}_{q_{j-1},q_j}$ for all $j \in [d]$. The strongest of these assumptions appears to be for $j = 1$ (because it involves the smallest "error rate" $1/q_1$), and this is certainly the case when relying on our reduction from LWE to LWR. For example, when using moduli $q_j = q^{d-j+1}$ where $q = n^{\omega(1)}$ is just slightly superpolynomial in n, the strongest assumption is the hardness of $\mathsf{LWR}_{q^{d+1},q^d}$, which involves an inverse error rate of $q^d = n^{\omega(d)} = n^{\omega(\log L)}$, where L is the PRF input length. This is a significantly weaker security guarantee than the one we get when using the domain-preserving synthesizers $T_{n,q,p}$.

3.4.3 Efficiency

To implement the synthesizer $T_{n,q,p}$ via an arithmetic procedure, we can use any fast matrix multiplication algorithm (e.g., Strassen's). In the case of the ring-based synthesizer $T_{R,q,p}$, we can multiply ring elements (in the standard power basis) in $O(n \log n)$ scalar operations mod q (see, e.g., [62]). In a practical parallel implementation, we can compute a matrix multiplication in the natural way using a size-$O(n^2)$, depth-2 arithmetic circuit over $\mathbb{Z}_q$, where the first layer of multiplication gates have fan-in 2 and the second layer of addition gates has fan-in n. The same is true for a product of ring elements in R_q, since it can be expressed as a matrix-vector product: multiplication by any fixed element $a \in R_q$ is a linear transformation. The gains in efficiency obtained by using a ring-based construction will be much more pronounced in the case of the degree-k synthesizers described in Chapter 4. We discuss these gains in detail in Section 4.2.1.

We remark that Naor and Reingold [76] describe several nice optimizations and additional features of their synthesizer-based PRFs, including compression of the secret key and faster amortized computation for a sequence of related inputs. Our functions are amenable to all these techniques as well.

3.4.4 Parallelism

We now show that our synthesizers are in $\mathrm{TC}^0 \subseteq \mathrm{NC}^1$ (i.e., they can be computed by constant-depth, polynomial-sized circuits of threshold gates with unbounded fan-in), and that the corresponding PRFs are therefore in $\mathrm{TC}^1 \subseteq \mathrm{NC}^2$. These results match the best constructions from [76].

Theorem 3.4.5. *If the bijection $K\colon \mathbb{Z}_p^m \to \mathbb{Z}_q^n$ is in TC^0, then so is $T_{n,q,p}$, and similarly for $T_{R,q,p}$ when the bijection $K\colon R_p^m \to R_q$ is in TC^0.*

We note that a wide class of natural bijections K are in TC^0. For instance, the "change of base" bijection described in Footnote 1 is in TC^0 because it can be

computed as a multisum of scalar products with the fixed powers of p, and both multisums and binary scalar products are in TC^0 [87]. In the ring setting, we can apply in parallel any TC^0 bijection from $\mathbb{Z}_q^m$ to $\mathbb{Z}_q$ to the coefficients of the input ring elements (in some arbitrary basis of the ring) to get a TC^0 bijection from R_p^m to R_q.

Proof of Theorem 3.4.5. The function $T_{n,q,p}$ can be computed as a matrix product modulo q, followed by parallel applications of $\lfloor \cdot \rceil_p$ to the entries of the matrix, followed by parallel applications of K. We claim that each of these three stages are in TC^0. Since this is true for K by hypothesis, we only need to analyze the other operations.

First, we note that the matrix product consists of m^2 parallel inner products of n-dimensional vectors, which each are a multisum of n parallel scalar products in $\mathbb{Z}_q$. The rounding step simply amounts to dropping some of the least-significant bits if q and p are both powers of two, or more generally, multiplying by $p/q \in \mathbb{Q}$ (with sufficient precision) and truncating the fractional part. Since modular multiplication and multisums are in TC^0 [87], the claim follows.

In the ring-based synthesizer $T_{R,q,p}$, instead of a matrix product over $\mathbb{Z}_q$ we have an outer product of vectors over R_q. Since ring multiplication is linear in each of its arguments, it is a special case of matrix multiplication. The translations from a ring element to its corresponding matrix and back are fixed linear transformations over $\mathbb{Z}_q$, so they also can be computed in TC^0. □

Finally, to analyze the parallel depth of our synthesizer-based PRFs from Construction 3.4.2, we define a single function $\mathbf{F}$ that evaluates any member of the family $\mathcal{F}^{(d)}$ (for any values of n and d) given the index and input, i.e.,

$$\mathbf{F}(\{\mathbf{S}_{i,b}\}, x) := F_{\{\mathbf{S}_{i,b}\}}(x)$$

where $\mathbf{S}_{i,b} \in \mathbb{Z}_q^{n \times m}$ for $i \in [2^d]$ and $b \in \{0,1\}$, and $x \in \{0,1\}^{2^d}$. Similarly, we define an analogous function $\mathbf{RF}(\{\mathbf{s}_{i,b}\}, x) = RF_{\{\mathbf{s}_{i,b}\}}(x)$ for the ring-based family $\mathcal{RF}^{(d)}$. Since the PRF families from Construction 3.4.4 compose the synthesizers in a binary tree

of depth d, where d is at most logarithmic in the total input length, we immediately have the following corollary of Theorem 3.4.5 above.

Corollary 3.4.6. The functions **F** and **RF** are in $\mathrm{TC}^1 \subseteq \mathrm{NC}^2$.

3.5 Open Problems and Related Work

The quasipolynomial moduli and inverse error rates used in our LWE-based security proofs are comparable to those used in recent fully homomorphic encryption (FHE) schemes (e.g., [34, 93, 25, 24, 22]), hierarchical identity-based encryption (HIBE) schemes (e.g., [28, 1, 2]), and other lattice-based constructions. However, there appears to be a major difference between our use of such strong assumptions, and that of schemes such as FHE/HIBE in the public-key setting. Constructions of the latter systems actually reveal LWE samples having very small error rates (which are needed to ensure correctness of decryption) to the attacker, and the attacker can break the cryptosystems by solving those instances. Therefore, the underlying assumptions and the true security of the schemes are essentially equivalent. In contrast, LWR, and the subsequent PRF, uses (small) errors *only as part of a thought experiment* in the security proof, not for any purpose in the operation of the function itself. This leaves open the possibility that these functions (or slight variants) remain secure even for much larger input lengths and smaller moduli than our proofs require. We conjecture that this is the case, even though we have not yet found security proofs (under standard assumptions) for these more efficient parameters. Certainly, determining whether there are effective cryptanalytic attacks is a very interesting and important research direction.

Therefore, the big open problem remains the possibility that $\mathsf{LWR}_{n,q,p}$ is indeed exponentially hard for $p = \mathrm{poly}(\lambda)$ and sufficiently large integers $q/p = \mathrm{poly}(\lambda)$. Towards this end, Alwen *et al.* [5] prove the security of LWR for q/p polynomial in λ, but also linearly dependent on the number of queries to the LWR-oracle in the security

proof. They do this by relying on a result [40] which shows a "lossy mode" for LWE. However, the dependence of the q/p ratio on the number of adversary queries in the proof results in it not being useful in the synthesizer construction. Also, this proof technique does not provide an equivalent meaningful statement about RLWR.

Our derandomization technique and LWR problem require working with moduli q greater than 2. This raises the question of there being an effective derandomization technique when the modulus $q = 2$. That is; is there an efficient, parallel PRF family based on the learning parity with noise (LPN) problem?

Most closely related to the techniques in this section are two results of Brakerski and Vaikuntanathan [24] and a follow-up work of Brakerski, Gentry, and Vaikuntanathan [22] on fully homomorphic encryption from LWE. In particular, the former work includes a "modulus reduction" technique for LWE-based cryptosystems, which maps a large-modulus ciphertext to a small-modulus one; this induces a shallower decryption circuit and allows the system to be "bootstrapped" into a fully homomorphic scheme using the techniques of [34]. The modulus-reduction technique involves a rounding operation much like the one we use to derandomize LWE; while they use it on ciphertexts that are already "noisy," we apply it to noise-free LWE samples.

CHAPTER IV

DIRECT CONSTRUCTION OF PSEUDORANDOM FUNCTIONS

In this chapter, we present another, potentially more efficient construction of a pseudorandom function family whose security is based on the intractibility of the LWE problem.

4.1 Overview

One moderate drawback of the synthesizer based function from Section 3.4 in Chapter 3 is that it involves $\lg k$ levels of rounding operations, which appears to lower-bound the depth of any circuit computing the function by $\Omega(\lg k)$. (The GGM based construction from Section 3.3 has k levels of rounding, and is therefore even worse.) Is it possible to do better?

Recall that in later works, Naor and Reingold [77] and Naor, Reingold, and Rosen [78] gave direct, more efficient number-theoretic PRF constructions which, while still requiring exponentiation in large multiplicative groups, can in principle be computed in very shallow circuit classes like NC^1 or even TC^0. Their functions can be interpreted as "degree-k" (or k-argument) synthesizers for arbitrary $k = \mathrm{poly}(\lambda)$, which immediately yield k-bit PRFs without requiring any composition. With this in mind, a natural question is whether there are direct LWE/LWR-based synthesizers of degree $k > 2$.

In this chapter, we give a positive answer to this question. Much like the functions of [77, 78], ours have a subset-product structure. We have public moduli $q \gg p$, and the secret key is a set of k matrices $\mathbf{S}_i \in \mathbb{Z}_q^{n \times n}$ (whose distributions may not necessarily

be uniform; see below) for $i = 1, \ldots, k$, along with a uniformly random $\mathbf{a} \in \mathbb{Z}_q^n$.[1] The function $F = F_{\mathbf{a},\{\mathbf{S}_i\}} \colon \{0,1\}^k \to \mathbb{Z}_p^n$ is defined as the "rounded subset-product"

$$F_{\mathbf{a},\{\mathbf{S}_i\}}(x_1 \cdots x_k) = \left\lfloor \mathbf{a} \cdot \prod_{i=1}^{k} \mathbf{S}_i^{x_i} \right\rceil_p . \tag{4.1.2}$$

The ring variant is analogous, replacing $\mathbf{a}$ with uniform $a \in R_q$ and each $\mathbf{S}_i$ by some $s_i \in R_q$ (or R_q^*, the set of multiplicatively invertible elements modulo q). This function is particularly efficient to evaluate using the discrete Fourier transform, as is standard with ring-based primitives (see, e.g., [61, 62]). In addition, similarly to [77, 78], one can optimize the subset-product operation via pre-processing, and evaluate the function in TC^0. We elaborate on these optimizations in Section 4.2.1.

For the security analysis of this construction, we have meaningful security proofs under various conditions on the parameters and computational assumptions, including standard LWE. In our LWE-based proof, two important issues are the distribution of the secret key components $\mathbf{S}_i$, and the choice of moduli q and p. For the former, it turns out that our proof needs the $\mathbf{S}_i$ matrices to be *short*, i.e., their entries should be drawn from the LWE error distribution. (LWE is no easier to solve for such short secrets [6].) This appears to be an artifact of our proof technique, which can be viewed as a variant of our LWE-to-LWR reduction, enhanced to handle adversarial queries. Summarizing the approach, define

$$G(x) = G_{\mathbf{a},\{\mathbf{S}_i\}}(x) := \mathbf{a} \cdot \prod_i \mathbf{S}_i^{x_i}$$

to be the subset-product function inside the rounding operation of (4.1.2). The fact that $F = \lfloor G \rceil_p$ lets us imagine adding *independent error terms* to each distinct output of G, but *only as part of a thought experiment* in the proof. More specifically, we consider a related *randomized* function $\tilde{G} = \tilde{G}_{\mathbf{a},\{\mathbf{S}_i\}} \colon \{0,1\}^k \to \mathbb{Z}_q^n$ that computes the subset-product by multiplying by each $\mathbf{S}_i^{x_i}$ in turn, but then also adds a fresh error

[1] To obtain longer function outputs, we can replace $\mathbf{a} \in \mathbb{Z}_q^n$ with a uniformly random matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ for any $m = \mathrm{poly}(n)$.

term immediately following each multiplication. Using the LWE assumption and induction on k, we can show that the randomized function $\tilde{G}$ is itself pseudorandom (over $\mathbb{Z}_q$), hence so is $\lfloor \tilde{G} \rceil_p$ (over $\mathbb{Z}_p$). Moreover, we show that for every queried input, with high probability $\lfloor \tilde{G} \rceil_p$ coincides with $\lfloor G \rceil_p = F$, because G and $\tilde{G}$ differ only by a cumulative error term that is small relative to q—this is where we need to assume that the entries of $\mathbf{S}_i$ are *small*. Finally, because $\lfloor \tilde{G} \rceil_p$ is a (randomized) pseudorandom function over $\mathbb{Z}_p$ that coincides with the deterministic function F on all queries, we can conclude that F is pseudorandom as well.

In the above-described proof strategy, the gap between G and $\tilde{G}$ grows *exponentially* in k, because we add a separate noise term following each multiplication by an $\mathbf{S}_i$, which gets enlarged when multiplied by all the later $\mathbf{S}_i$. So in order to ensure that $\lfloor \tilde{G} \rceil_p = \lfloor G \rceil_p$ on all queries, our LWE-based proof needs both the modulus q and inverse error rate $1/\alpha$ to exceed $\lambda^{\Omega(k)}$. In terms of efficiency and security, this compares rather unfavorably with the quasipolynomial $\lambda^{\Omega(1)}$ bound in the proof for our tree-based construction, though on the positive side, the direct degree-k construction has better circuit depth. However, just as with the synthesizer construction in Section 3.1.2, it is unclear whether such strong assumptions and large parameters are actually *necessary* for security, or whether the matrices $\mathbf{S}_i$ really need to be short.

In particular, it would be nice if the direct PRF construction were secure if the $\mathbf{S}_i$ matrices were *uniformly random* over $\mathbb{Z}_q^{n \times n}$, because we could then recursively compose the function in a k-ary tree to rapidly extend its input length.[2] It would be even better to have a security proof for a smaller modulus q and inverse error rate $1/\alpha$, ideally both polynomial in λ even for large k. While we have been unable to find such a security proof under standard LWE, we do give a very tight proof under a new, interactive "*related samples*" LWE/LWR assumption. Roughly speaking, the

[2]Note that we can always compose the degree-k function with our degree-2 synthesizers from above, but this would only yield a tree with 2-ary internal nodes.

assumption says that LWE/LWR remains hard even when the sampled $\mathbf{a}_i$ vectors are related by adversarially chosen subset-products of up to k given random matrices (drawn from some known distribution). This provides some evidence that the function may indeed be secure for appropriately distributed $\mathbf{S}_i$, small modulus q, and large k. For full details see Section 4.2.4.

4.2 Direct PRF Construction

*Construction 4.2.1 ((Ring-)*LWE *degree-k PRF).* For parameters $\lambda \in \mathbb{N}$, moduli $q \geq p \geq 2$, positive integers $m, n = \text{poly}(\lambda)$, and input length $k \geq 1$, the family $\mathcal{F}$ consists of functions from $\{0,1\}^k$ to $\mathbb{Z}_p^{m \times n}$. A function $F \in \mathcal{F}$ is indexed by some $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ and $\mathbf{S}_i \in \mathbb{Z}^{n \times n}$ for each $i \in [k]$, and is defined as

$$F(x) = F_{\mathbf{A},\{\mathbf{S}_i\}}(x_1 \cdots x_k) := \left\lfloor \mathbf{A} \cdot \prod_{i=1}^{k} \mathbf{S}_i^{x_i} \right\rceil_p .$$

We endow $\mathcal{F}$ with the distribution where $\mathbf{A}$ is chosen uniformly at random, and below we consider a number of natural distributions for the $\mathbf{S}_i$.

The ring-based family $\mathcal{RF}$ is defined similarly to consist of functions from $\{0,1\}^k$ to R_p, where we replace $\mathbf{A}$ with uniformly random $a \in R_q$ and each $\mathbf{S}_i$ with some $s_i \in R$.

4.2.1 Efficiency

Using arithmetic circuits, matrix products and the rounding function can be computed in a fashion similar to the one detailed in Section 3.4.3, and this enables us to construct a function in the family $\mathcal{F}$ from Construction 4.2.1. The ring variant of Construction 4.2.1 appears to be more efficient to evaluate. Consider a function $F \in \mathcal{RF}$ as in Construction 4.2.1. As is standard with ring-based primitives (see, e.g., [61, 62]), one could store the ring elements $a, s_1, \ldots, s_k$ as vectors in $\mathbb{Z}_q^n$ using the discrete Fourier transform or "Chinese remainder" representation modulo q (that

is, by evaluating a and the s_i as polynomials at the n roots of $z^n + 1$ modulo q), so that multiplication of two ring elements just corresponds to a coordinate-wise product of their vectors. Then to evaluate the function, one would just compute a subset-product of the appropriate vectors, then interpolate the result to the power-basis representation, using essentially an n-dimensional Fast Fourier Transform over $\mathbb{Z}_q$, in order to perform the rounding operation. For the interesting case of $k = \omega(\log n)$, the sequential runtime of this method is dominated by the kn scalar multiplications in $\mathbb{Z}_q$ to compute the subset-product; in parallel, the arithmetic depth (over $\mathbb{Z}_q$) is $O(\log(nk))$. Alternatively, the subset-product part of the function might be computed even faster by storing the *discrete logs*, with respect to some arbitrary generator g of $\mathbb{Z}_q^*$, of the Fourier coefficients of a and s_i.[3] The subset-product then becomes a subset-sum, followed by exponentiation modulo q, or even just a table lookup if q is relatively small. Assuming that additions mod $q - 1$ are significantly less expensive than multiplications mod q, the sequential runtime of this method is dominated by the $O(n \log n)$ scalar operations in the FFT, and the parallel arithmetic depth is again $O(\log n)$.

4.2.2 Parallelism

We now consider the depth of functions from families $\mathcal{F}$ and $\mathcal{RF}$ from Construction 4.2.1 when implemented via boolean circuits with threshold gates. As in Section 3.4.3, we define a single function **RF** that evaluates any function in the function family $\mathcal{RF}$ (for any values of n and k) given the index and input as follows:

$$\mathbf{RF}(a, \{s_i\}, x) := F_{a,\{s_i\}}(x),$$

where $a \in R_q$, $s_i \in R$ for $i \in [k]$ and $x \in \{0,1\}^k$.

Theorem 4.2.2. *The function* **RF** *is in* $TC^0 \subseteq NC^1$.

[3] If necessary, one would also store binary mask vectors indicating which Fourier coefficients are zero, and hence not in $\mathbb{Z}_q^*$.

We thus establish that the ring-based construction matches (asymptotically) the shallowest known PRFs based on the DDH and factoring problems [77, 78]. This is also an improvement over the depth of ring-based PRF constructed from pseudorandom synthesizers in Section 3.4, as shown in Corollary 3.4.6.

Proof. If the inputs $a, \{s_i\}$ to the function **RF** are stored as vectors in $\mathbb{Z}_q^n$ using the discrete Fourier transform or the "Chinese Remainder" representation, the PRF computation is a series of three steps: the coordinate-wise multi-product of the vectors, and the n-dimensional Fast Fourier Transform over $\mathbb{Z}_q$ to convert the result into its coefficient representation in R_q, finally followed by the coordinate-wise rounding step, as detailed in Section 4.2.1 above. It thus suffices to prove that each of these three operations are in TC^0.

The coordinate-wise multi-product of the vectors can be performed as n modular multi-products in parallel, and this is in TC^0 [87]. The same reference also shows that Fast Fourier Transform over $\mathbb{Z}_q$ is also in TC^0. The rounding step is also in TC^0, as shown in the proof of Theorem 3.4.5. □

Recall that computing a function $F \in \mathcal{F}$ involves a subset-product of matrices. Generally speaking, matrix multi-product does not appear to be in TC^0 (if it were, then TC^0 would equal NC^1 [64]). However, in our case the matrices are known in advance (the variable input is the subset), so it may be possible to reduce the depth of the computation via preprocessing, using ideas from [87]. As described in Section 3.4.3, both binary matrix product and rounding can be implemented in TC^0, so at worst the function **F**, analogous to the family $\mathcal{F}$ from Construction 4.2.1, is in TC^1 by computing the subset product in a tree-like fashion, followed by a final rounding step.

4.2.3 Security Proof Under LWE

Our first theorem says that when the entries of the $\mathbf{S}_i$ are "small," i.e., chosen from a suitable LWE error distribution, the degree-k construction is a PRF under a suitable LWE assumption.

Theorem 4.2.3. *Let $\chi = D_{\mathbb{Z},r}$ for some $r > 0$, and let $q \geq p \cdot k(Cr\sqrt{n})^k \cdot \lambda^{\omega(1)}$ for a suitable universal constant C. Endow the family $\mathcal{F}$ from Definition 4.2.3 with the distribution where each $\mathbf{S}_i$ is drawn independently from $\chi^{n\times n}$. Then assuming the hardness of decision-$\mathsf{LWE}_{n,q,\chi}$, the family $\mathcal{F}$ is pseudorandom.*

An analogous theorem holds for the ring-based family $\mathcal{RF}$, under decision-RLWE.

Theorem 4.2.4. *Let χ be the distribution over the ring R where each coefficient (with respect to the power basis) is chosen independently from $D_{\mathbb{Z},r}$ for some $r > 0$, and let $q \geq p \cdot k(r\sqrt{n} \cdot \omega(\sqrt{\log n}))^k \cdot \lambda^{\omega(1)}$. Endow the family $\mathcal{RF}$ from Definition 4.2.3 with the distribution where each s_i is drawn independently from χ. Then assuming the hardness of decision-$\mathsf{RLWE}_{n,q,\chi}$, the family $\mathcal{RF}$ is pseudorandom.*

We first prove Theorem 4.2.3 for the standard LWE construction. To aid the proof, it helps to define a family $\mathcal{G}$ of functions, which are simply the unrounded counterparts of the functions in $\mathcal{F}$.

Definition 4.2.5. *For parameters n, q, m, k as in Construction 4.2.1, the family $\mathcal{G}$ consists of functions $G\colon \{0,1\}^k \to \mathbb{Z}_q^{n\times n}$. A function $G \in \mathcal{G}$ is indexed by some $\mathbf{A} \in \mathbb{Z}_q^{m\times n}$ and $\mathbf{S}_i \in \mathbb{Z}^{n\times n}$ for $i \in [k]$, we define*

$$G_{\mathbf{A},\{\mathbf{S}_i\}}(x_1 \cdots x_k) := \mathbf{A} \cdot \prod_{i=1}^{k} \mathbf{S}_i^{x_i}.$$

We endow $\mathcal{G}$ with the same distribution over $\mathbf{A}$ and the $\mathbf{S}_i$ as $\mathcal{F}$ has; i.e., $\mathbf{A}$ is chosen uniformly at random and $\mathbf{S}_i$ are chosen from the error distribution $\chi^{n\times n}$ from the statement of Theorem 4.2.3.

Proof of Theorem 4.2.3. We proceed via a sequence of games, much like in the proof of Theorem 3.2.2. First as a "thought experiment" we define a new family $\tilde{\mathcal{G}}$ of functions from $\{0,1\}^k$ to $\mathbb{Z}_q^{m \times n}$. This family is a counterpart to $\mathcal{G}$, but with two important differences: it is a PRF family *without* any rounding (and hence, with rounding as well), but each function in the family has an exponentially large key. Alternatively, one may think of the functions in $\tilde{\mathcal{G}}$ as *randomized* functions with small keys. Then we show that with overwhelming probability, the rounding of $\tilde{G} \leftarrow \tilde{\mathcal{G}}$ agrees with the rounding of the corresponding $G \in \mathcal{G}$ on all the attacker's queries, because the outputs of the two functions are relatively close. It follows that the rounding of $G \leftarrow \mathcal{G}$ (i.e., $F \leftarrow \mathcal{F}$) cannot be distinguished from a uniformly random function, as desired.

More formally, we define the following games:

Game H_0. This is the real PRF attack game against the family $\mathcal{F}$: we choose an $F \leftarrow \mathcal{F}$ (so $F(\cdot) = \lfloor G(\cdot) \rceil_p$ for $G \leftarrow \mathcal{G}$), and the attacker has oracle access to $F(\cdot)$.

Game H_1. Here we instead choose $\tilde{G} \leftarrow \tilde{\mathcal{G}}$, where the family $\tilde{\mathcal{G}}$ is given in Definition 4.2.6 below. The choice of $\tilde{G}$ induces a corresponding $G \in \mathcal{G}$ having the same distribution as in H_0. (This is simply because the key of G is just a portion of the key of $\tilde{G}$.) To be precise, we choose $\tilde{G}$ "lazily" as the attacker makes queries, because the description of $\tilde{G}$ has exponential size; see the remarks following Definition 4.2.6 for details.

The attacker has oracle access to $\lfloor \tilde{G}(\cdot) \rceil_p$, but with one exception: on query x, define the "bad event" Bad_x for that query to be

$$\left\lfloor \tilde{G}(x) + [-B, B]^{m \times n} \right\rceil_p \neq \{ \lfloor \tilde{G}(x) \rceil_p \},$$

where $B = k(Cr\sqrt{n})^k$ is a constant as chosen in the statement of Lemma 4.2.7. That is, Bad_x indicates whether any entry of $\tilde{G}(x) \in \mathbb{Z}_q^{m \times n}$ is "too close" to another element

of $\mathbb{Z}_q$ that rounds to a different value in $\mathbb{Z}_p$. Note that because $q \gg p \cdot B$, a value $y \in \mathbb{Z}_q$ is "too close" in this sense if and only if $\lfloor y - B \rceil_p \neq \lfloor y + B \rceil_p$, so Bad_x can be efficiently detected given only the value of $\tilde{G}(x)$. If Bad_x occurs any of the attacker's queries, then the game immediately aborts.

In Lemma 4.2.7 below, we show that for every fixed $x \in \{0,1\}^k$, with overwhelming probability over the choice of $\tilde{G} \leftarrow \tilde{\mathcal{G}}$ and the induced $G \in \mathcal{G}$, it is the case that $G(x) \in \tilde{G}(x) + [-B, B]^{m \times n} \bmod q$. Hence $\lfloor G(x) \rceil_p = \lfloor \tilde{G}(x) \rceil_p$ so long as Bad_x does not occur, and the attacker's queries are answered exactly as they are in H_0, subject to the game not aborting. It follows that for any (potentially unbounded) attacker $\mathcal{A}$,

$$\mathbf{Adv}_{H_0,H_1}(\mathcal{A}) \leq \Pr[\text{some } \mathsf{Bad}_x \text{ occurs in } H_1 \text{ with attacker } \mathcal{A}] + \mathrm{negl}(\lambda). \quad (4.2.3)$$

We do not directly bound the probability that some Bad_x occurs in H_1, but instead defer to the analysis of the next game, where we can show that it is indeed negligible.

Game H_2. Here we choose U to be a uniformly random function from $\{0,1\}^k$ to $\mathbb{Z}_q^{m \times n}$ (defined "lazily" as the attacker makes queries). The attacker has oracle access to $\lfloor U(\cdot) \rceil_p$, with the same "bad event" and abort condition as in H_1, but defined relative to U instead of $\tilde{G}$.

In Theorem 4.2.8 below, we show that under the LWE assumption from the theorem statement, no efficient adversary can distinguish (given oracle access) between $\tilde{G} \leftarrow \tilde{\mathcal{G}}$ and a uniformly random function U. Because the Bad_x event in H_1 (respectively, H_2) for a query x can be tested efficiently given query access to $\tilde{G}$ (resp., U), a trivial simulation implies that for any efficient attacker $\mathcal{A}$, we have $\mathbf{Adv}_{H_1,H_2}(\mathcal{A}) \leq \mathrm{negl}(\lambda)$. For the same reasons, it also follows by a straightforward simulation that for any efficient attacker $\mathcal{A}$,

$$|\Pr[\text{some } \mathsf{Bad}_x \text{ occurs in } H_1 \text{ with } \mathcal{A}] - \Pr[\text{some } \mathsf{Bad}_x \text{ occurs in } H_2 \text{ with } \mathcal{A}]| \leq \mathrm{negl}(\lambda).$$

In H_2, because U is a uniformly random function, for any particular query x the probability that Bad_x occurs is bounded by $(2B+1) \cdot p/q = \mathrm{negl}(\lambda)$, by assumption on q. By a union bound over all $\mathrm{poly}(\lambda)$ queries of an efficient $\mathcal{A}$, and then applying Equation (4.2.3), we therefore have that

$$\Pr[\text{some } \mathsf{Bad}_x \text{ occurs in } H_1 \text{ with } \mathcal{A}] = \mathrm{negl}(\lambda) \quad \Rightarrow \quad \mathbf{Adv}_{H_0,H_1}(\mathcal{A}) = \mathrm{negl}(\lambda).$$

Game H_3. Here we still choose a uniformly random function U and give the attacker oracle access to $\lfloor U(\cdot) \rceil_p$. For each query x we define the event Bad_x as in game H_2, but still answer the query and continue with the game even if Bad_x occurs. From the above analysis of H_2 it follows that for any (potentially unbounded) attacker $\mathcal{A}$ making $\mathrm{poly}(\lambda)$ queries, we have

$$\mathbf{Adv}_{H_2,H_3}(\mathcal{A}) \leq \Pr[\text{some } \mathsf{Bad}_x \text{ occurs in } H_2 \text{ with } \mathcal{A}] = \mathrm{negl}(\lambda).$$

Finally, observe that $\lfloor U(\cdot) \rceil_p$ is a truly random function from $\{0,1\}^k$ to $\mathbb{Z}_p^{m \times n}$, up to the bias involved in rounding the uniform distribution on $\mathbb{Z}_q$ to $\mathbb{Z}_p$. Because $q \geq p \cdot \lambda^{\omega(1)}$, this bias is negligible (and there is no bias if p divides q).

By the triangle inequality, it follows that for any efficient $\mathcal{A}$, we have $\mathbf{Adv}_{H_0,H_3}(\mathcal{A}) = \mathrm{negl}(\lambda)$, and this completes the proof. □

We now define the family $\tilde{\mathcal{G}}$ used in the proof of Theorem 4.2.3.

Definition 4.2.6. *For parameters n, q, m, k as in Construction 4.2.1, and error distribution χ over $\mathbb{Z}$ as in Theorem 4.2.3, the family $\tilde{\mathcal{G}}^{(i)}$ for $0 \leq i \leq k$ is defined inductively to consist of functions from $\{0,1\}^i$ to $\mathbb{Z}_q^{m \times n}$; we define $\tilde{\mathcal{G}} = \tilde{\mathcal{G}}^{(k)}$.*

- *For $i = 0$, a function $\tilde{G} \in \tilde{\mathcal{G}}^{(0)}$ is indexed by some $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$, and is defined simply as $\tilde{G}_{\mathbf{A}}(\varepsilon) = \mathbf{A}$. We endow $\tilde{\mathcal{G}}^{(0)}$ with the distribution where $\mathbf{A}$ is chosen uniformly at random.*

- *For $i \geq 1$, a function $\tilde{G} \in \tilde{\mathcal{G}}^{(i)}$ is indexed by some $\tilde{G}' \in \tilde{\mathcal{G}}^{(i-1)}$, plus an $\mathbf{S}_i \in \mathbb{Z}^{n \times n}$ and error matrices $\mathbf{E}_{x'} \in \mathbb{Z}^{m \times n}$ for each $x' \in \{0,1\}^{i-1}$ (where $\{0,1\}^0$ is the singleton set $\{\varepsilon\}$). For $x = (x', x_i) \in \{0,1\}^i$ where $|x'| = i - 1$, the function is defined as*

$$\tilde{G}(x) = \tilde{G}_{\tilde{G}', \mathbf{S}_i, \{\mathbf{E}_{x'}\}}(x', x_i) := \tilde{G}'(x') \cdot \mathbf{S}_i^{x_i} + x_i \cdot \mathbf{E}_{x'} \bmod q.$$

We endow $\tilde{\mathcal{G}}^{(i)}$ with the distribution where $\tilde{G}' \leftarrow \tilde{\mathcal{G}}^{(i-1)}$, and all the entries of $\mathbf{S}_i$ and every $\mathbf{E}_{x'}$ are chosen independently from χ.

Note that a function $\tilde{G} \in \tilde{\mathcal{G}}$ is fully specified by $\mathbf{A}$, $\{\mathbf{S}_i\}_{i \in [k]}$, and *exponentially* (in k) many error matrices $\mathbf{E}_{x_1 \cdots x_{i-1}}$ for all $x \in \{0,1\}^k$ and $i \in [k]$; these error matrices are what prevents $\tilde{\mathcal{G}}$ itself from being used as a PRF family. However, as needed in the proof of Theorem 4.2.3 (game H_1), the error matrices can be chosen "lazily," since the value of $\tilde{G}(x)$ depends only on $\mathbf{A}$, $\{\mathbf{S}_i\}$, and $\mathbf{E}_{x_1 \cdots x_{i-1}}$ for $i \in [k]$. For a function $\tilde{G} = \tilde{G}_{\mathbf{A}, \{\mathbf{S}_i\}, \{\mathbf{E}_{x'}\}} \in \tilde{\mathcal{G}}$, we define its *induced* function in the family $\mathcal{G}$ to be $G = G_{\mathbf{A}, \{\mathbf{S}_i\}}$. Note that for $\tilde{G} \leftarrow \tilde{\mathcal{G}}$, the induced function G has the same marginal distribution as if it had been chosen from $\mathcal{G}$ directly.

The following lemma is used in the analysis of game H_1.

Lemma 4.2.7. Let $x \in \{0,1\}^k$ be arbitrary. Then except with $2^{-\Omega(n)} = \mathrm{negl}(\lambda)$ probability over the choice of $\tilde{G} = \tilde{G}_{\mathbf{A}, \{\mathbf{S}_i\}, \{\mathbf{E}_{x'}\}} \leftarrow \tilde{\mathcal{G}}$ and its induced function $G = G_{\mathbf{A}, \{\mathbf{S}_i\}} \in \mathcal{G}$, we have

$$G(x) \in \tilde{G}(x) + [-B, B]^{m \times n} \bmod q$$

for some $B = k \cdot (Cr\sqrt{n})^k$, where C is a universal constant.

Note that it is in this lemma that we require the entries of the seed matrices $\mathbf{S}_i$ to be drawn from a subgaussian distribution.

Proof of Lemma 4.2.7. Observe that

$$\begin{aligned}\tilde{G}(x_1 \cdots x_k) &= (\cdots((\mathbf{A} \cdot \mathbf{S}_1^{x_1} + x_1 \cdot \mathbf{E}_\varepsilon) \cdot \mathbf{S}_2^{x_2} + x_2 \cdot \mathbf{E}_{x_1}) \cdots) \cdot \mathbf{S}_k^{x_k} + x_k \cdot \mathbf{E}_{x_1 \cdots x_{k-1}} \bmod q \\ &= \underbrace{\mathbf{A} \cdot \prod_{i=1}^{k} \mathbf{S}_i^{x_i}}_{G(x)} + x_1 \cdot \mathbf{E}_\varepsilon \cdot \prod_{i=2}^{k} \mathbf{S}_i^{x_i} + x_2 \cdot \mathbf{E}_{x_1} \cdot \prod_{i=3}^{k} \mathbf{S}_i^{x_i} + \cdots + x_k \cdot \mathbf{E}_{x_1 \cdots x_{k-1}} \bmod q.\end{aligned}$$

Now by Lemma 2.1.2, except with probability $2^{-\Omega(n)}$, for every $i \in [k]$ we have $s_1(\mathbf{S}_i) \le O(r\sqrt{n})$ and $\|\mathbf{e}\| \le O(r\sqrt{n})$ for every row $\mathbf{e}$ of the error matrices $\mathbf{E}_{x_1 \cdots x_{i-1}}$. Therefore, each row of the k cumulative error matrices $\mathbf{E}_{x_1 \cdots x_{i-1}} \cdot \prod_{j=i+1}^{k} \mathbf{S}_j^{x_j}$ (for $i \in [k]$) has Euclidean length at most $O(r\sqrt{n})^k$, and so its entries are bounded by the same quantity in magnitude. The claim follows. □

Theorem 4.2.8. *Under the* LWE *assumption from the statement of Theorem 4.2.3, the family $\tilde{\mathcal{G}}$ of functions from $\{0,1\}^k$ to $\mathbb{Z}_q^{m \times n}$ is pseudorandom for any $k = \mathrm{poly}(\lambda)$.*

In the proof we will need the following intermediate function families.

Definition 4.2.9. *For n, q, m, and χ as in Construction 4.2.1, and an integer $i \ge 1$, the family $\mathcal{H}^{(i)}$ consists of functions from $\{0,1\}^i$ to $\mathbb{Z}_q^{m \times n}$. A function H from the family is indexed by some $\mathbf{S}_i \in \mathbb{Z}^{n \times n}$ and matrices $\mathbf{A}_{x'} \in \mathbb{Z}_q^{m \times n}, \mathbf{E}_{x'} \in \mathbb{Z}^{m \times n}$ for each $x' \in \{0,1\}^{i-1}$ (where $\{0,1\}^0 = \{\varepsilon\}$). It is defined as*

$$H(x) = H_{\mathbf{S}_i, \{\mathbf{A}_{x'}\}, \{\mathbf{E}_{x'}\}}(x', x_i) := \mathbf{A}_{x'} \cdot \mathbf{S}_i^{x_i} + x_i \cdot \mathbf{E}_{x'} \bmod q,$$

where $|x'| = i - 1$. We endow $\mathcal{H}$ with the distribution where each $\mathbf{A}_{x'}$ is uniformly random and independent, and all the entries of $\mathbf{S}_i$ and $\mathbf{E}_{x'}$ are chosen independently from χ. We remark that an $H \leftarrow \mathcal{H}^{(i)}$ can be chosen "lazily" in the natural way.

Proof of Theorem 4.2.8. We prove that each family $\tilde{\mathcal{G}}^{(i)}$ is pseudorandom by induction on i, from 0 to k. The base case of $i = 0$ is trivial by construction. For $i \ge 1$, we prove the claim by the following series of games.

Game H_0. We (lazily) choose a $\tilde{G} \leftarrow \tilde{\mathcal{G}}^{(i)}$ and give the attacker oracle access to $\tilde{G}(\cdot)$.

Game H_1. We (lazily) choose an $H \leftarrow \mathcal{H}^{(i)}$ (defined above) and give the attacker oracle access to $H(\cdot)$.

We claim that $H_0 \overset{c}{\approx} H_1$ under the inductive hypothesis that $\tilde{\mathcal{G}}^{(i-1)}$ is a PRF family. To prove this, we design an efficient simulator $\mathcal{S}$ that is given oracle access to a function $F\colon \{0,1\}^{i-1} \to \mathbb{Z}_q^{m \times n}$, where F is either $\tilde{G}' \leftarrow \tilde{\mathcal{G}}^{(i-1)}$ or a uniformly random function, and $\mathcal{S}$ emulates either game H_0 or H_1 (respectively) to an attacker. The simulator $\mathcal{S}$ first chooses an $\mathbf{S}_i \leftarrow \chi^{n \times n}$, and on each query $x = (x', x_i)$ from the attacker where $|x'| = i - 1$, $\mathcal{S}$ queries its oracle to get $\mathbf{A}_{x'} = F(x')$, chooses an $\mathbf{E}_{x'} \leftarrow \chi^{m \times n}$ (if it has not already been defined by a previous query), and returns $\mathbf{A}_{x'} \cdot \mathbf{S}_i^{x_i} + x_i \cdot \mathbf{E}_{x'}$ to the attacker. It is clear by the definitions of $\tilde{\mathcal{G}}^{(i)}$ and $\mathcal{H}^{(i)}$ that if F is some $\tilde{G}' \leftarrow \tilde{\mathcal{G}}^{(i-1)}$, then $\mathcal{S}$ emulates access to $\tilde{G}_{\tilde{G}', \mathbf{S}_i, \{\mathbf{E}_{x'}\}} \in \tilde{\mathcal{G}}^{(i)}$ with the appropriate distribution, whereas if F is a uniformly random function, then $\mathcal{S}$ emulates access to $H \leftarrow \mathcal{H}^{(i)}$.

Game H_2. We (lazily) choose a uniformly random function $U\colon \{0,1\}^i \to \mathbb{Z}_q^{m \times n}$ and give the attacker oracle access to $U(\cdot)$.

We claim that $H_1 \overset{c}{\approx} H_2$ under the decision-LWE assumption from Theorem 4.2.3. To prove this, we design an efficient simulator $\mathcal{S}$ that is given access to an oracle $\mathcal{O}$ that outputs arbitrarily many pairs $(\mathbf{A}, \mathbf{B}) \in \mathbb{Z}_q^{m \times n} \times \mathbb{Z}_q^{m \times n}$, drawn either as a group of samples $(\mathbf{A}, \mathbf{B} = \mathbf{AS} + \mathbf{E} \bmod q)$ from the LWE distribution $A_{\mathbf{S},\chi}$ (for the same $\mathbf{S} \leftarrow \chi^{n \times n}$) or from the uniform distribution, and $\mathcal{S}$ emulates either game H_1 or H_2 (respectively) to an attacker. Under the decision-LWE assumption, this will establish the claim. The simulator $\mathcal{S}$ answers queries $x = (x', x_i)$ where $|x'| = i - 1$ in the following way: if x' has never been queried before, then it draws a new sample $(\mathbf{A}_{x'}, \mathbf{B}_{x'})$ from $\mathcal{O}$ and stores it, otherwise it looks up the already stored $(\mathbf{A}_{x'}, \mathbf{B}_{x'})$.

It then returns $\mathbf{A}_{x'}$ if $x_i = 0$, and $\mathbf{B}_{x'}$ if $x_i = 1$. It is clear by inspection and the definition of $\mathcal{H}^{(i)}$ that $\mathcal{S}$ has the claimed behavior given the two types of oracles $\mathcal{O}$.

By the triangle inequality, we have $H_0 \overset{c}{\approx} H_2$, i.e., $\tilde{\mathcal{G}}^{(i)}$ is a pseudorandom function family. This finishes the induction and the proof. □

We remark that in the induction used in the above proof, the total number of hybrid games obtained by "unrolling" the inductive hypotheses is only linear in k, because the proof invokes the inductive hypothesis exactly once, in the transition from game H_0 to H_1. Therefore, the total distinguishing advantage between $\tilde{G} \leftarrow \tilde{\mathcal{G}}^{(k)}$ and a uniformly random function can grow only linearly (in k) relative to the LWE distinguishing advantage.

We now analyze the ring-LWE construction.

Proof sketch for Theorem 4.2.4. The proof proceeds almost identically to the proof of Theorem 4.2.3, so we only outline the few small differences. We define the function families $\mathcal{RG}$ and $\mathcal{R}\tilde{\mathcal{G}}$ in exactly the same fashion as the families $\mathcal{G}$ and $\tilde{\mathcal{G}}$, respectively, with $a \in R_q$, $s_i \in R$ and $e_{x'} \in R$ substituting $\mathbf{A}$, $\mathbf{S}_i$ and $\mathbf{E}_{x'}$ respectively. In the games, the bad event Bad_x occurs if any *coefficient* of $R\tilde{G}(x) \in R_q$ (for $R\tilde{G} \leftarrow \mathcal{R}\tilde{\mathcal{G}}$) is "too close" to another element in $\mathbb{Z}_q$ having a different rounded value, where "too close" is defined using the interval $[-B, B]$ for $B = k(r\sqrt{n} \cdot \omega(\sqrt{\log n}))^k/\sqrt{n}$. For this bound B, the analogue of Lemma 4.2.7 (which bounds the cumulative error terms, i.e., the difference $R\tilde{G}(x) - RG(x)$) follows immediately from Lemma 2.1.3. Finally, pseudorandomness of the family $\mathcal{R}\tilde{\mathcal{G}}$ follows analogously to the proof of Theorem 4.2.8, via families $\mathcal{RH}^{(i)}$ defined similarly to $\mathcal{H}^{(i)}$. □

Remark 4.2.10. By almost identical proofs, a similar subset-product-like construction

$$F_{\mathbf{A},\{\mathbf{S}_{i,b}\}}(x_1 \cdots x_k) = \left\lfloor \mathbf{A} \cdot \prod_{i=1}^{k} \mathbf{S}_{i,x_i} \right\rceil_p,$$

for uniform $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ and matrices $\mathbf{S}_{i,b} \in \mathbb{Z}^{n \times n}$ (for $i \in [k], b \in \{0,1\}$), and the analogous function in the ring setting, are also PRF families for the same parameters and distributions as in Theorem 4.2.3 and Theorem 4.2.4. (These functions are analogous to the factoring-based PRF of [78].) While the secret keys are about twice as large as their counterparts' from Construction 4.2.1, these functions are more "symmetric," which may be important in practice (e.g., to prevent timing attacks).

4.2.4 Security Proof Under Interactive LWR

We now present an "interactive" LWR assumption and prove that under this assumption, the degree-k construction from Construction 4.2.1 is a PRF under an appropriate distribution of the $\mathbf{S}_i$. The advantage of this proof is that it allows us to prove security for a small modulus q and inverse error rate (both small polynomials in λ), and it also works for uniformly random (or uniform invertible) matrices $\mathbf{S}_i$, among other distributions. For example, this allows us to compose the degree-k construction with itself (or with any other PRF) in a k-ary tree in a synthesizer-like style as outlined in Section 3.4. The drawback to our proof is that it relies on a stronger assumption that is harder to evaluate or falsify, because it allows the adversary to make queries to its challenger.

Definition 4.2.11 (k-subset-product LWR). *Let $q \geq p$ be integer moduli. We describe a pair of games, which are parameterized by integers $k \geq 1$ and $m, n = \mathrm{poly}(\lambda)$, and a distribution ψ over $\mathbb{Z}_q^{n \times n}$ (e.g., the uniform distribution). In both games, we choose $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ uniformly at random and $\mathbf{S}_i \leftarrow \psi$ independently for each $i \in [k]$, then give $\mathbf{A}$ and $\mathbf{S}_i$ for $i \in [k-1]$ to the attacker. We then allow the attacker to adaptively make queries to a function $H \colon \{0,1\}^{k-1} \to \mathbb{Z}_p^{m \times n}$. In the first game, the function H is defined to be*

$$H(x) := \left\lfloor \mathbf{A} \cdot \prod_{i=1}^{k-1} \mathbf{S}_i^{x_i} \cdot \mathbf{S}_k \right\rceil_p ;$$

in the second game, H is a uniformly random function. The k-subset-product LWR problem*, denoted* k-$\mathsf{LWR}_{q,p,n,m,\psi}$*, is to distinguish between these two games with an advantage non-negligible in* λ*. The* k*-subset-product ring-*LWR *problem is defined analogously. (A subset-product version of (ring-)*LWE *is also easy to formulate, where instead of rounding we add random and independent error terms to each answer.)*

We make a few simple observations about the k-LWR problem. First note that $\mathbf{B}_x = \mathbf{A} \cdot \prod_{i=1}^{k-1} \mathbf{S}_i^{x_i}$ is the part of the product that changes for each new query. Since $\mathbf{A}$ and all the $\mathbf{S}_i$ for $i \in [k-1]$ are given to the attacker, it can compute each $\mathbf{B}_x$ on its own, and its goal is to determine whether the challenger is returning rounded products $\lfloor \mathbf{B}_x \cdot \mathbf{S}_k \rceil_p$ or uniformly random and independent values. In effect, the k-LWR problem is therefore to solve LWR when the sampled $\mathbf{A}$ matrices are related by adversarially chosen subset-products of given random matrices $\mathbf{S}_i$. To avoid an efficient attack (as outlined in the introduction), the distribution ψ should be chosen so that the product of many $\mathbf{S}_i \leftarrow \psi$ does not significantly reduce the entropy of $\mathbf{A} \prod_i \mathbf{S}_i$. It appears that restricting ψ to invertible elements is most effective for this purpose.

We also observe that 1-$\mathsf{LWR}_{q,p,n,m,\psi}$ is just the standard $\mathsf{LWR}_{n,q,p}$ problem given m samples, where the secret matrix $\mathbf{S}$ is chosen from ψ. The problems form a hierarchy over k, that is, k-$\mathsf{LWR}_{q,p,n,m,\psi}$ no harder than $(k-1)$-$\mathsf{LWR}_{q,p,n,m,\psi}$, by a reduction that just prepends 0 to all queries, and withholds $\mathbf{S}_1$ from the attacker.

Theorem 4.2.12. *Endow the family $\mathcal{F}$ from Construction 4.2.1 with the distribution where each $\mathbf{S}_i$ is drawn from some distribution ψ. Then, assuming that k-$\mathsf{LWR}_{q,p,n,m,\psi}$ problem is hard, the family $\mathcal{F}$ is pseudorandom.*

Unlike our inductive proof of Theorem 4.2.8, which transitions from the PRF family to a random function by "dropping" the secret key components $\mathbf{S}_i$ from $i = 1$ to k, the proof of Theorem 4.2.12 drops them from $i = k$ down to 1. This prevents

the error terms from growing with k (because the errors are not compounded by multiplication with other $\mathbf{S}_i$), which is what allows us to use a small modulus q if we so desire. However, this style of proof also seems to require an interactive assumption, so that a simulator can answer queries involving the component $\mathbf{S}_i$ that is being dropped between adjacent games.

Proof of Theorem 4.2.12. We prove this by induction over k. For $k = 0$, the claim follows trivially by construction. For $k \geq 1$, we again proceed via a series of games.

Game H_0. This is the real PRF attack game against the family $\mathcal{F}$: we choose an $F \leftarrow \mathcal{F}$, and the attacker has oracle access to $F(\cdot)$.

Game H_1. We choose $F \leftarrow \mathcal{F}$. For attacker queries of the form $x = x_1 \dots x_{k-1}1$, we return uniformly random and independent value (consistent with prior answers), and for queries of the form $x = x_1 \dots x_{k-1}0$, we return $F(x) = \left\lfloor \mathbf{A} \cdot \prod_{i=1}^{k-1} \mathbf{S}_i^{x_i} \right\rceil_p$.

We claim that $H_0 \overset{c}{\approx} H_1$ by a straightforward reduction assuming the hardness of k-LWR. As proof, we construction a simulator $\mathcal{S}$ that interacts with an oracle $\mathcal{O}$ that implements one of the two games from the k-LWR problem, and emulates either H_0 or H_1 respectively. The simulator is first given some matrices $\mathbf{A}$ and $\mathbf{S}_i$ for $i \in [k-1]$. It then answers attacker queries $x = (x', 0) \in \{0,1\}^k$ by returning $\left\lfloor \mathbf{A} \cdot \prod_{i=1}^{k-1} \mathbf{S}_i^{x_i} \right\rceil_p$, and answers queries $x = (x', 1) \in \{0,1\}^k$ by returning $\mathcal{O}(x')$ to the attacker. It is clear by inspection that the behavior of $\mathcal{S}$ is as claimed.

Game H_2. We lazily choose a uniformly random function $U \colon \{0,1\}^k \to \mathbb{Z}_p^{m \times n}$ and give the attacker oracle access to $U(\cdot)$.

We claim that $H_1 \overset{c}{\approx} H_2$ by the inductive hypothesis. This is because in game H_1, queries ending in 1 are already answered uniformly, while queries ending in 0 are answered according to a function drawn from the family $\mathcal{F}$ of degree $(k-1)$. This

family is pseudorandom by the inductive hypothesis, and the fact that $(k-1)$-LWR is no easier than k-LWR.

This completes the induction and the proof. □

4.3 Open Problems and Related Work

As with the discussion in Section 3.5, the major question guiding further research seems to be the huge exponential moduli and inverse error rates used in the LWE-based security proofs. As noted earlier, we need the (small) random errors only in the security proofs as part of a thought experiment. Again, it is our belief that smaller (polynomial in λ) parameters would also yield secure PRFs.

Note that in the PRF from Construction 4.2.1, if we draw the secret key components from the uniform (or error) distribution and allow k to be too large relative to q, then the function can become insecure via a simple attack (and our new "interactive" LWR assumption, which yields a tight security proof, becomes false). This is easiest to see for the ring-based function: representing each $s_i \in R_q$ by its vector of "Fourier coefficients" over $\mathbb{Z}_q^n$, each coefficient is 0 with probability about $1/q$ (depending on the precise distribution of s_i). Therefore, with noticeable probability the product of $k = O(q \log n)$ random s_i will have all-0 Fourier coefficients, i.e., will be $0 \in R_q$. In this case our function will return zero on the all-1s input, in violation of the PRF requirement. (A similar but more complicated analysis can also be applied to the matrix-based function.) Of course, an obvious countermeasure is just to restrict the secret key components to be *invertible*; to our knowledge, this does not appear to have any drawback in terms of security. In fact, it is possible to show that the decision-(ring-)LWE problem remains hard when the secret is restricted to be invertible (and otherwise drawn from the uniform or error distribution), and this fact may be useful in further analysis of the function with more efficient parameters.

In summary, some of the concrete questions raised in this chapter are:

- Is there a security proof for the direct degree-k PRF construction from this chapter (with $k = \omega(1)$) for poly(λ)-bounded moduli and inverse error rates, under a non-interactive assumption?

- For the direct degree-k PRF construction from this chapter, is there a security proof (under a non-interactive assumption) for uniformly random $\mathbf{S}_i$? Is there any provable security advantage to using *invertible* $\mathbf{S}_i$?

Along with the related work pointed out in Section 3.5, we note that the "somewhat homomorphic" cryptosystem from [24] that supports degree-k operations (along with all prior ones, e.g., [34, 93]) involves an inverse error rate of $\lambda^{O(k)}$, much like the LWE-based proof in Section 4.2.3.

Building on the modulus reduction technique of [24], Brakerski *et al.* [22] showed that homomorphic cryptosystems can support certain degree-k functions using a much smaller modulus and inverse error rate of $\lambda^{O(\log k)}$. The essential idea is to interleave the homomorphic operations with several "small" modulus-reduction steps in a tree-like fashion, rather than performing all the homomorphic operations followed by one "huge" modulus reduction. This very closely parallels the difference between the construction from this chapter and the Naor-Reingold-like [76] composed synthesizer defined in Chapter 4. Indeed, we found Construction 4.2.1 earlier and the result of [22] inspired our search for a PRF having similar tree-like structure and quasipolynomial error rates. Given our degree-2 synthesizer from Chapter 3, the solution turned out to largely be laid out in the work of [76]. We find it very interesting that the same quantitative phenomena arise in two seemingly disparate settings (PRFs and FHE).

CHAPTER V

KEY-HOMOMORPHIC PSEUDORANDOM FUNCTIONS

In this chapter, we give constructions of LWE-based PRFs that improve on all prior constructions in the following senses – they are based on weaker LWE assumptions, are much more efficient in time and space, and are still highly parallel. In addition, these PRFs are *key-homomorphic*, which make them useful for additional applications. Remarkably, some of our RLWE based constructions have key sizes, public parameters, and incremental runtimes on consecutive inputs are all quasi-linear $\tilde{O}(\lambda)$ in the security parameter λ, which is optimal up to polylogarithmic factors. To our knowledge, these are the first low-depth PRFs (whether key homomorphic or not) enjoying any of these efficiency measures together with nontrivial proofs of 2^λ security under any conventional assumption.

5.1 Overview

A pseudorandom function family $\mathcal{F} = \{F_s\}$ is *key homomorphic* if the set of keys has a group structure and if there is an efficient algorithm that, given $F_s(x)$ and $F_t(x)$ (but not s or t), outputs $F_{s+t}(x)$. Naor, Pinkas, and Reingold [74] constructed, in the random oracle model, a very simple key-homomorphic PRF family based on the decisional Diffie-Hellman problem, and gave applications like distributing the operation of a Key Distribution Center. Boneh *et al.* [18] constructed the first key-homomorphic PRFs *without* random oracles, and described many more applications (all of which are very efficient in their use of the PRF), including symmetric-key proxy re-encryption, updatable encryption, and PRFs secure against related-key attacks (cf. [13, 53]). The construction of Boneh *et al.* is LWE-based, and builds upon ideas used in the *non*-key-homomorphic LWE-based PRFs of Chapter 4. Apart from the

obvious added functionality of being key-homomorphic, the Boneh *et al.* construction improves upon Construction 4.2.1 in a very crucial way – its key-size is *independent* of the length of the input, unlike Construction 4.2.1, the key-size of which is linear in the input length k.

One drawback of the construction and proof from [18] is its rather strong LWE assumption that it inherits from Construction 4.2.1 (which can be implemented in $TC^0 \subseteq NC^1$ and, by consequence, large parameters and runtimes. For example, to obtain a PRF of input length λ with exponential 2^λ provable security against known lattice attacks, the secret keys and public parameters respectively need to be at least λ^3 and λ^6 bits, and the runtime to evaluate the function is at least λ^7 bit operations (to produce λ^2 output bits), not counting some polylogarithmic $\log^{O(1)} \lambda$ factors. However, the synthesizer-based Construction 3.4.4 (in $TC^1 \subseteq NC^2$) and sequential GGM-based one from Chapter 3 can be proved secure under much weaker LWE assumptions, and hence can have much better parameters and runtimes. A natural question, therefore, is whether there exist *key-homomorphic* PRFs with similar security and efficiency characteristics.

In this chapter, we answer the above question in the affirmative, by giving new constructions of key-homomorphic PRFs that have substantially better efficiency, and still enjoy very high parallelism. As compared with [18], we improve the key size from λ^3 to λ bits, the public parameters from λ^6 to λ^2 bits, and the runtime from λ^7 to $\lambda^{\omega+1}$ bit operations (always omitting $\log^{O(1)} \lambda$ factors), where $\omega \in [2, 2.373]$ is the exponent of matrix multiplication.

We also give even more efficient key-homomorphic PRFs based on the *ring*-LWE problem [62, 63]. Compared with the ring-based analogue of [18], and again ignoring $\log^{O(1)} \lambda$ factors, here our keys and public parameters are only λ bits (improving upon λ^3 and λ^4, respectively), and the runtime is only λ^2 bit operations to produce λ output bits (from λ^5 to produce λ^2). In addition, the *incremental* computation of our

PRF on successive inputs (e.g., in a counter-like mode) has runtime only λ. Functions having these parameters can be implemented in $TC^1 \subseteq NC^2$, though seemingly not in TC^0 or NC^1. See Table 2 for a full comparison with the PRFs in Chapters 3 and 4 and also [18].

Table 2: Example instantiations of our key-homomorphic PRF (for input length λ and provable 2^λ security against the best known lattice algorithms) as compared with prior lattice-based PRFs. "KH" denotes whether the construction is key homomorphic, while "Expan" and "Sequen" are respectively the expansion and sequentiality (as defined in Equations (5.2.4), (5.2.8)) of the tree T used in the instantiation (or, for prior constructions, their close analogues). Omitting polylogarithmic $\log^{O(1)} \lambda$ factors, "Key" and "Params" are respectively the bit lengths of the secret key and public parameters; "Time/Out" is the best known runtime (in bit operations) per output bit, where $\omega \in [2, 2.373]$ is the exponent of matrix multiplication; and "Out" is the output length in bits. The quantities in brackets refer to the corresponding ring-based constructions.

Reference	KH?	Expan	Sequen	Key	Params	Time/Out	Out
Ch. 3 (GGM)	N	1	λ	λ [λ]	λ^2 [λ]	λ^2 [λ]	λ [λ]
Ch. 3 (synth)	N	$\log_2 \lambda$	$\log_2 \lambda$	λ^3 [λ^2]	0 [0]	$\lambda^{\omega-1}$ [λ]	λ^2 [λ]
Chapter 4	N	λ	1	λ^5 [λ^3]	0 [0]	λ^4 [λ^2]	λ^2 [λ^2]
[18]	Y	$\lambda - 1$	1	λ^3 [λ^3]	λ^6 [λ^4]	λ^5 [λ^3]	λ^2 [λ^2]
Chapter 5	Y	1	$\lambda - 1$	λ [λ]	λ^2 [λ]	λ^ω [λ]	λ [λ]
Chapter 5	Y	$\log_4 \lambda$	$\log_4 \lambda$	λ [λ]	λ^2 [λ]	λ^ω [λ]	λ [λ]

To our knowledge, these are the first *low-depth* PRFs (whether key homomorphic or not) having nontrivial proofs of exponential 2^λ security under any conventional assumption along with *quasi-optimal* $\tilde{O}(\lambda)$ key sizes or incremental runtimes, or quasilinear $\tilde{O}(\lambda)$ nonincremental runtime per output bit. For example, the GGM construction [39] can have small keys and quasilinear nonincremental runtime per output bit (using a quasi-optimal PRG), but it is highly sequential. The Naor-Reingold constructions [76, 77], which are highly parallel, have at least quadratic λ^2 key sizes and runtime per output bit, even assuming exponential security of the

underlying hard problems. And factoring-based constructions [78] fare much worse due to subexponential-time factoring algorithms.

In their parallelism and underlying LWE assumptions, our functions are qualitatively very similar to the synthesizer- and GGM-based ones from Chapter 3 (see Table 2); however, the constructions and proofs are completely different. Instead, our construction can be seen as a substantial generalization of the one of Boneh *et al.* [18], in that theirs is an instantiation of ours with a linear-depth "left spine" tree. By contrast, our construction can be securely instantiated with *any* binary tree, thanks to a new proof technique that may be of use elsewhere. The shape of the tree determines the final parameters and parallelism of the resulting function: roughly speaking, its "left depth" determines the strength of the LWE assumption in the proof, while its "right depth" determines its parallelism. Interestingly, a complete binary tree turns out to be *very far from optimal* for the parameters we care about. Optimal trees can be found efficiently using dynamic programming, and provide input lengths that are roughly the *square* of those yielded by complete binary trees.

5.2 Construction and Analysis

In this section we define and analyze our key-homomorphic PRF, and compare it with prior LWE-based constructions from Chapters 3 and 4 above, and [18]. The construction involves various parameters (e.g., matrix dimension n, modulus q, tree T) which are all chosen so that the algorithms are polynomial-time in the security parameter λ, as usual. As in prior work, we work in a model where the PRF family is defined with respect to some *random public parameters* that are known to all parties, including the adversary. These parameters may be generated by a trusted party, or by the user along with the secret key.

For a full (but not necessarily complete) binary tree T—i.e., one in which every non-leaf node has two children—let $|T|$ denote the number of its leaves. If $|T| \geq 1$ (i.e.,

T is not the empty tree), let $T.l, T.r$ respectively denote the left and right subtrees of T (which may be empty trees).

For the reader's convenience, we point out that it will be useful to recall the *gadget* matrix and the *bit-decomposition* operation [70]. We present the needed background in Section 2.4.

We now define our function families.

Definition 5.2.1. *Given matrices $\mathbf{A}_0, \mathbf{A}_1 \in \mathbb{Z}_q^{n \times n\ell}$ and a full binary tree T of at least one node, define the function $\mathbf{A}_T \colon \{0,1\}^{|T|} \to \mathbb{Z}_q^{n \times n\ell}$ recursively as*

$$\mathbf{A}_T(x) = \begin{cases} \mathbf{A}_x & \text{if } |T| = 1 \\ \mathbf{A}_{T.l}(x_l) \cdot \mathbf{G}^{-1}(\mathbf{A}_{T.r}(x_r)) & \text{otherwise,} \end{cases}$$

where in the second case we parse $x = x_l \| x_r$ for $x_l \in \{0,1\}^{|T.l|}, x_r \in \{0,1\}^{|T.r|}$.

Construction 5.2.2 (Key-Homomorphic PRF). The function family

$$\mathcal{F}_{\mathbf{A}_0, \mathbf{A}_1, T, p} = \left\{ F_{\mathbf{s}} \colon \{0,1\}^{|T|} \to \mathbb{Z}_p^{n\ell} \right\}$$

is parameterized by matrices $\mathbf{A}_0, \mathbf{A}_1 \in \mathbb{Z}_q^{n \times n\ell}$, a binary tree T, and a modulus $p \leq q$, which may all be considered public parameters. A member of the family is indexed by some $\mathbf{s} \in \mathbb{Z}_q^n$, and is defined as

$$F_{\mathbf{s}}(x) := \lfloor \mathbf{s} \cdot \mathbf{A}_T(x) \rceil_p.$$

For security based on LWE, we take $\mathbf{A}_0, \mathbf{A}_1$ and the secret key $\mathbf{s}$ to be uniformly random over $\mathbb{Z}_q$; see Theorem 5.2.3 below for a formal security statement. Similarly to LWE, it may also be possible to prove security when the entries of $\mathbf{s}$ are drawn from the LWE error distribution (see [6]). However, most applications of key-homomorphic PRFs need to use uniformly random secret keys anyway, so we do not pursue this question further.

Because rounding is nearly linear, i.e., $\lfloor a+b \rceil_p = \lfloor a \rceil_p + \lfloor b \rceil_p + e$ for some $e \in \{0, \pm 1\}$, it is easy to see that the family $\mathcal{F}_{\mathbf{A}_0,\mathbf{A}_1,T,p}$ defined above is "almost" additively key homomorphic, as defined in [18]. That is, for any keys $F_{\mathbf{s}}, F_{\mathbf{t}}$ in the family, we have

$$F_{\mathbf{s}+\mathbf{t}}(x) = F_{\mathbf{s}}(x) + F_{\mathbf{t}}(x) + \mathbf{e},$$

where $\|\mathbf{e}\|_\infty \leq 1$. As long as the entries of the error term $\mathbf{e}$ are sufficiently smaller than the output modulus p, this near-homomorphism is sufficient for all the applications described in [18], and for obtaining security against related-key attacks [53].

Notice that the vast majority of the cost of computing $F_{\mathbf{s}}(x)$ is in computing $\mathbf{A}_T(x)$, which can done "publicly" without any knowledge of the secret key $\mathbf{s}$.[1] This property can be very important for the efficiency of certain applications, such as the *homomorphic* evaluation of $F_{\mathbf{s}}$ given an encryption of $\mathbf{s}$. In addition, notice that if $\mathbf{A}_T(x)$ has been computed and all the intermediate matrices saved, then $\mathbf{A}_T(x')$ can be *incrementally* computed much more efficiently for an x' that differs from x in just a single bit. Specifically, one only needs to recompute the matrices for the internal nodes of T on the path from the leaf corresponding to the changed bit to the root. As in the related constructions from Chapter 4.2 and [18], this can significantly speed up successive evaluations of $F_{\mathbf{s}}$ on related inputs, e.g., in a counter-like mode using a Gray code.

Relation to [18]. Our key-homomorphic PRF can be viewed as a substantial generalization of the one of Boneh *et al.* [18]. Specifically, their construction can be obtained from ours by instantiating it with a tree T that consists of a "left spine" with leaves for all its right children. Because all the right subtrees are just leaves, the only matrices ever decomposed with $\mathbf{G}^{-1}$ are $\mathbf{A}_0$ and $\mathbf{A}_1$. Therefore, we can replace

[1] For a few choices of the tree T, it can be faster to compute $\mathbf{s} \cdot \mathbf{A}_T(x)$ left-to-right without explicitly computing $\mathbf{A}_T(x)$, but such trees are rare and yield bad parameters.

them in the public parameters by the binary matrices $\mathbf{B}_b = \mathbf{G}^{-1}(\mathbf{A}_b)$, yielding the construction $F_{\mathbf{r}}(x) = \left\lfloor \mathbf{r} \cdot \prod_{i=1}^{|x|} \mathbf{B}_{x_i} \right\rceil_p$ from [18].[2]

The use of a "left-spine" tree T (as in [18]) yields an instantiation which is *maximally parallel*—in our language (defined below), it has *sequentiality* $s(T) = 1$. The major drawback is that it also has maximal *expansion* $e(T) = |T| - 1$. In our security theorem (Theorem 5.2.3 below), the LWE approximation factor and modulus q grow *exponentially* with $e(T)$, so using a tree with large expansion leads to a very strong hardness assumption, and therefore large secret keys and public parameters. By contrast, using trees T with better expansion-sequentiality tradeoffs allows us obtain much better key sizes and efficiency. See the discussion in the following subsections and Table 2 for further details.

5.2.1 Security

In our security proof, the modulus q and underlying LWE error rate, and hence also the dimension n needed to obtain a desired level of provable security, are largely determined by a certain parameter of the tree T which we call the *expansion* $e(T)$. Essentially, the expansion is the maximum number of terms of the form $\mathbf{G}^{-1}(\cdot)$ that are ever consecutively multiplied together when we unwind the recursive definition of $\mathbf{A}_T$, or $\mathbf{A}_{T'}$ for related trees T' considered in the security proof. Formally, the expansion of T is defined by the recurrence

$$e(T) = \begin{cases} 0 & \text{if } |T| = 1 \\ \max\{e(T.l) + 1\ ,\ e(T.r)\} & \text{otherwise.} \end{cases} \tag{5.2.4}$$

This is simply the "left depth" of the tree, i.e., the maximum length of a root-to-leaf path, counting only edges from parents to their left children.

[2]Here we have ignored the small detail that in our construction, the matrix $\mathbf{A}_{x_1}$ corresponding to the leftmost leaf in the tree is not decomposed, so our instantiation is actually $F_{\mathbf{s}}(x) = \lfloor \mathbf{s} \cdot \mathbf{A}_{x_1} \cdot \prod_{i=2}^{|x|} \mathbf{B}_{x_i} \rceil$. However, it is easy to verify that in the construction of [18], the secret key may be of the form $\mathbf{r} = \mathbf{sG}$ for some $\mathbf{s} \in \mathbb{Z}_q^n$. Then $\mathbf{rB}_{x_1} = \mathbf{sA}_{x_1}$, which corresponds to our construction.

We can now state our main security theorem.

Theorem 5.2.3. *Let T be any full binary tree, χ be some distribution over $\mathbb{Z}$ that is subgaussian with parameter $r > 0$ (e.g., a bounded or discrete Gaussian distribution with expectation zero), and*

$$q \geq p \cdot r\sqrt{|T|} \cdot (n\ell)^{e(T)} \cdot \lambda^{\omega(1)}.$$

Then over the uniformly random and independent choice of $\mathbf{A}_0, \mathbf{A}_1 \in \mathbb{Z}_q^{n \times n\ell}$, the family $\mathcal{F}_{\mathbf{A}_0,\mathbf{A}_1,T,p}$ with secret key chosen uniformly from $\mathbb{Z}_q^n$ is a secure PRF family, under the decision-$\mathsf{LWE}_{n,q,\chi}$ assumption.

An outline of the proof, which contains all the main and new ideas, is given in Section 5.3.1; the formal proof appears in in Section 5.3.2.

Notice that the modulus-to-noise ratio for the underlying LWE problem is $q/r \approx (n \log q)^{e(T)}$, i.e., exponential in the expansion $e(T)$. Known reductions [86, 80, 23] (for $r \geq 3\sqrt{n}$) guarantee that such an LWE instantiation is at least as hard as (quantumly) approximating various lattice problems in the worst case to within $\approx q/r$ factors on n-dimensional lattices. Known algorithms for achieving such factors take time exponential in $n/\log(q/r) = \tilde{\Omega}(n/e(T))$, so in order to obtain provable 2^λ security against the best known lattice algorithms, the best parameters we can use are

$$n = e(T) \cdot \tilde{\Theta}(\lambda) \quad \text{and} \quad \log q = e(T) \cdot \tilde{\Theta}(1). \tag{5.2.7}$$

These parameters determine the runtimes and key sizes of the construction, as analyzed below.

We conclude this discussion of security by reiterating a point we made before in Sections 3.5 and 4.3, that, as in the constructions from Chapter 4 and [18], and in contrast with essentially all lattice-based *encryption* schemes, it is possible that our PRF is actually secure for *much smaller* parameters than our proof requires. For

example, taking $q = \text{poly}(n)$ even for large $e(T)$, with $p|q$ to ensure that rounding produces "unbiased" output, may actually be secure—but we do not know how to prove it. (We also do not know of any effective attacks against such parameters.) The reason for this possibility is that the function itself does not actually expose any low-error-rate LWE samples to the attacker; they are used only in the proof as part of a thought experiment. Whether any of the constructions from this chapter, Chapter 4 or [18] can be proved secure for smaller parameters under a standard assumption is a fascinating open question. For the remainder of the paper, we deal only with parameters for which we can *prove* security under (ring-)LWE.

5.2.2 Size, Time, and Depth

Here we briefly analyze the secret key and public parameter sizes, runtime, and circuit depth of our PRFs, always normalizing to 2^λ provable security under standard lattice assumptions. In some cases these quantities are not very practical (or even asymptotically good), especially when the tree T has large expansion. In Section 5.2.4 we give a much more efficient construction using ring-LWE, which can be quasi-optimal in key size, public parameters, and depth (simultaneously).

The secret key, which is a uniformly random element of $\mathbb{Z}_q^n$, has size $\Theta(n \log q)$, which is $e(T)^2 \cdot \tilde{\Theta}(\lambda)$ by Equation (5.2.7). The public parameters, being two $n \times n\ell$ matrices over $\mathbb{Z}_q$, are $\Theta(n^2 \log^2 q) = e(T)^4 \cdot \tilde{\Theta}(\lambda^2)$ bits.

For runtime, computing $\mathbf{A}_T(x)$ from scratch takes one decomposition with $\mathbf{G}^{-1}$ and one $(n \times n\ell)$-by-$(n\ell \times n\ell)$ matrix multiplication over $\mathbb{Z}_q$ per internal node of T. (As mentioned above, incremental computation of $\mathbf{A}_T(x)$ on related inputs can be much faster.) Using naïve matrix multiplication, this is a total of $\Theta(|T| \cdot n^3 \log^2 q)$ ring operations in $\mathbb{Z}_q$, which translates to $e(T)^6 \cdot \tilde{\Theta}(\lambda^4)$ bit operations by Equation (5.2.7) (even using quasi-linear-time multiplication in $\mathbb{Z}_q$, which is needed only when $\log q \neq \tilde{O}(1)$). This can be improved somewhat using asymptotically faster matrix multiplication,

but still remains a rather large $\Omega(|T| \cdot n^\omega \log^2 q)$, where $\omega \geq 2$ is the exponent of matrix multiplication.

For certain trees T our construction is highly parallelizable, i.e., it can be computed by a low-depth circuit. First, notice that each $\mathbb{Z}_q$-entry of $\mathbf{s} \cdot \mathbf{A}_T(x)$ (and hence each $\mathbb{Z}_p$-entry of the PRF output) can be computed independently. This is because each column of $\mathbf{A}_T(x)$ can be computed independently, by induction and the fact that $\mathbf{G}^{-1}$ works independently on the columns of $\mathbf{A}_{T.r}(x_r)$. Next, since linear operations over $\mathbb{Z}_q$ can be computed by depth-one arithmetic circuits (with unbounded fan-in), the circuit depth of our construction is proportional to the maximum nesting depth of $\mathbf{G}^{-1}(\cdot)$ expressions when we fully unwind the definition of $\mathbf{A}_T$. We call this the *sequentiality* $s(T)$ of the tree T, which is formally defined by the recurrence

$$s(T) = \begin{cases} 0 & \text{if } |T| = 1 \\ \max\{e(T.l)\ ,\ e(T.r) + 1\} & \text{otherwise.} \end{cases}$$

This is simply the "right depth" of the tree, i.e., the maximum length of a root-to-leaf path, counting only edges from parents to their right children.

5.2.3 Instantiations

Here we discuss some interesting instantiations of the tree T and the efficiency properties of the resulting functions; see Table 2 for a summary. Generally speaking, for a given tree size $|T|$ (the PRF input length) there is a tradeoff between expansion $e(T)$ and sequentiality $s(T)$. Flipping this around, given bounds e, s we are interested in obtaining a largest possible tree T such that $e(T) \leq e$ and $s(T) \leq s$; let $t(e, s)$ denote the size of such a tree. At first blush, it may be surprising that under the simplifying restriction $e = s$, a complete binary tree of depth s is *very far* from optimal! To see

this, notice that

$$t(e,s) = \begin{cases} 1 & \text{if } e = 0 \text{ or } s = 0 \\ t(e-1,s) + t(e,s-1) & \text{otherwise.} \end{cases}$$

The base cases follow from the fact that only a single leaf satisfies the bounds, and in the recursive case, the first and second terms respectively denote the sizes of the optimal left and right subtrees. It is easy to verify that this recurrence is simply the one that defines the *binomial coefficients*:

$$t(e,s) = \binom{e+s}{e} = \binom{e+s}{s}.$$

One can also efficiently construct an optimal tree for given e, s using dynamic programming.

For example, if we restrict to $e = s$, then by Stirling's approximation we get that $t(e,s) = \binom{2s}{s} \approx 4^s/\sqrt{s\pi}$. Said another way, we can get a PRF with input length $|T|$ where the expansion and sequentiality are both $\approx \log_4(|T|)$. By contrast, a complete binary tree with these parameters has size only $2^s \approx \sqrt{|T|}$. By Theorem 5.2.3 and Equation (5.2.7), this means we can get a PRF with input length λ and security 2^λ having sequentiality $O(\log \lambda)$ and secret keys of quasi-optimal bit length $\tilde{O}(\lambda)$.

By ignoring parallelism, one can reduce the expansion even further by letting T be a "right spine" with leaves for all its left children. Then $e(T) = 1$ and $s(T) = |T| - 1$, yielding even better parameters: the underlying LWE assumption has a nearly polynomial $n^{\omega(1)}$ approximation factor, and for security level 2^λ we still obtain secret keys of quasi-optimal bit length $\tilde{O}(\lambda)$; moreover, here the hidden factors are at least a $\log \lambda$ factor smaller than in the case above.

Illustrative Examples. We now present a few different instantiations of the tree T to illustrate the working of the PRF. In the rest of this subsection, we will use T_v to denote the tree rooted at node v.

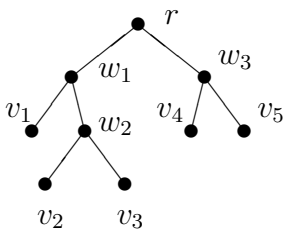

Figure 1: A typical instantiation – tree T_1

We first begin a typical tree T_1, displayed in Figure 1, . The root is labelled r, other internal nodes are labeled w_i, and leaf nodes are labeled v_i. By Definition 5.2.1, we expand $F_{\mathbf{s}}(x_1 \dots x_5)$, to see how the sequentiality is intrinsic to the evaluation of the PRF.

$$\begin{aligned} F_{\mathbf{s}}(x_1 \dots x_5) &= \lfloor \mathbf{s} \cdot \mathbf{A}_{T_r}(x_1 \dots x_5) \rceil_p \\ &= \left\lfloor \mathbf{s} \cdot \mathbf{A}_{T_{w_1}}(x_1 x_2 x_3) \cdot \mathbf{G}^{-1}(\mathbf{A}_{T_{w_3}}(x_4 x_5)) \right\rceil_p \\ &= \cdots \\ &= \left\lfloor \mathbf{s} \cdot \mathbf{A}_{x_1} \cdot \mathbf{G}^{-1}\big(\mathbf{A}_{x_2} \cdot \mathbf{G}^{-1}(\mathbf{A}_{x_3})\big) \cdot \mathbf{G}^{-1}\big(\mathbf{A}_{x_4} \mathbf{G}^{-1}(\mathbf{A}_{x_5})\big) \right\rceil_p \end{aligned}$$

The running time is dependent upon the bits x_3 and x_5, because the matrices they choose $\mathbf{A}_{x_3}$ and $\mathbf{A}_{x_5}$ are nested in two layers of bit-decomposition operations. This clearly corresponds to the right-depth of the leaves v_3 and v_5, which corresponds to the sequentiality of the tree T_1. Also to be noted is that the expansion of the tree is determined by the left-depth of the leaves v_1 and v_2, and is also two.

We also illustrate the Boneh *et al.* [18] construction tree and the tree which achieves the maximum number of leaves, and hence the maximum input length, for $e = s = 2$ in Figures 2 and 3 respectively.

5.2.4 Ring Variant

Due to the several matrix multiplications (of dimension at least n) involved in computing $\mathbf{A}_T(x)$, our LWE-based construction is not very practically efficient. Fortunately,

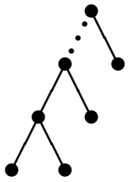

Figure 2: The [18] "left spine" tree

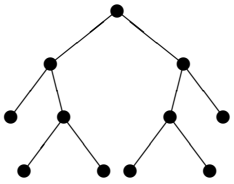

Figure 3: The "balanced tree" for $e = s = 2$

we can obtain a much more efficient analogue based on the ring-LWE problem [62]. Here we just describe the construction and analyze its efficiency. The proof of security based on ring-LWE proceeds in essentially the same way as the one for our main construction, and is therefore omitted.

Construction 5.2.4. Fix some *row* vectors $\mathbf{a}_0, \mathbf{a}_1 \in R_q^\ell$, and for a binary tree T, define $\mathbf{a}_T \colon \{0,1\}^{|T|} \to R_q^\ell$ recursively as

$$\mathbf{a}_T(x) = \begin{cases} \mathbf{a}_x & \text{if } |T| = 1 \\ \mathbf{a}_{T.l}(x_l) \cdot \mathbf{g}^{-1}(\mathbf{a}_{T.r}(x_r)) & \text{otherwise,} \end{cases}$$

where in the second case we parse $x = x_l \| x_r$ for $x_l \in \{0,1\}^{|T.l|}, x_r \in \{0,1\}^{|T.r|}$.

We define the function family

$$\mathcal{F}_{\mathbf{a}_0,\mathbf{a}_1,T,p} = \left\{ F_s \colon \{0,1\}^{|T|} \to R_p^\ell \right\},$$

which is parameterized by row vectors $\mathbf{a}_0, \mathbf{a}_1 \in R_q^\ell$, a binary tree T, and a modulus $p \leq q$. A member of the family is indexed by some $s \in R$ (or R_q), and is defined as

$$F_s(x) := \lfloor s \cdot \mathbf{a}_T(x) \rceil_p.$$

Analysis. Evaluating $\mathbf{a}_T(x)$ from scratch takes one decomposition with $\mathbf{g}^{-1}$ and one vector-matrix multiplication of dimension $\ell = \log q$ over R_q per internal node of T, for a total of $O(|T| \cdot \ell^2)$ ring operations in R_q. Ring operations in R_q can be performed in $O(n \log n)$ scalar operations over $\mathbb{Z}_q$, and $\mathbf{g}^{-1}$ can be computed in $O(n \log q)$ time. Using a tree T with expansion $e(T) = \tilde{O}(1)$, by Equation (5.2.7) we can get a PRF with input length λ and 2^λ security (under conventional assumptions) running in $\tilde{O}(\lambda^2)$ bit operations to output at least λ bits. When T has polylogarithmic depth, the incremental cost per invocation is reduced to $\tilde{O}(\lambda)$ bit operations, which is quasi-optimal.

As an optimization, and analogously to the LWE-based construction, each R_q-entry of $\mathbf{a}_T(x) \in R_q^\ell$ can be computed *independently* in $O(|T| \cdot \ell)$ ring operations each. Therefore, we can compute each R_p-entry of the output (yielding at least n output bits) in just $O(|T| \cdot \ell)$ ring operations. This may be useful in applications that do not need the entire large output length.

5.3 Security Proof

In this section we prove Theorem 5.2.3, which says that $F_{\mathbf{s}}(x) = \lfloor \mathbf{s} \cdot \mathbf{A}_T(x) \rceil_p$ from Construction 5.2.2 is a PRF under the LWE assumption, for appropriate parameters.

5.3.1 Proof Outline

We start with an overview of the proof, which highlights the central (new) ideas. (For technical reasons, the formal proof proceeds a bit differently than this outline, but the main ideas are the same.) The basic strategy, used in Chapters 3 and 4

above, is to define a sequence of hybrid games where the function inside the rounding operation $\lfloor \cdot \rceil_p$ changes in ways that are indistinguishable to the adversary, either statistically or computationally. As in Chapters 3 and 4, and [18], these changes include introducing small additive terms that are "rounded away" and hence preserve the input-output behavior (with high probability), and replacing LWE instances with uniformly random ones. In addition, we introduce a new proof technique described within.

Let T be any full binary tree, and suppose its leftmost leaf v is at depth $d > 1$. (If $d = 1$, then $|T| = 1$ and the function is trivially a PRF based on the "learning with rounding" problem, which is as hard as LWE for our choice of parameters, or even slightly better ones – see Chapter 3 and [5].) In the real attack game, the adversary has oracle access to $F_{\mathbf{s}}(\cdot)$, which, by unwinding the definition of $\mathbf{A}_T$, is of the form

$$F_{\mathbf{s}}(x) = \lfloor \mathbf{s} \cdot \mathbf{A}_T(x) \rceil_p = \left\lfloor \mathbf{s} \cdot \mathbf{A}_{x_0} \cdot \underbrace{\prod_{i=1}^{d} \mathbf{G}^{-1}(\mathbf{A}_{T_i}(x'_i))}_{\mathbf{S}_T(x')} \right\rceil_p ,$$

where subtree T_i is the right child of v's ith ancestor, and $x = x_0 \| x' = x_0 \| x'_1 \| \cdots \| x'_d$ where $|x_0| = 1$ and $|x'_i| = |T_i|$ for all i.

We next consider a hybrid game in which $\mathbf{s} \cdot \mathbf{A}_b$ for $b \in \{0, 1\}$ is replaced by an LWE vector $\mathbf{s} \cdot \mathbf{A}_b + \mathbf{e}_b$, for some short error vectors $\mathbf{e}_0, \mathbf{e}_1$. That is, the adversary instead has oracle access to the function

$$F'_{\mathbf{s},\mathbf{e}_0,\mathbf{e}_1}(x) := \lfloor (\mathbf{s} \cdot \mathbf{A}_{x_0} + \mathbf{e}_{x_0}) \cdot \mathbf{S}_T(x') \rceil_p = \lfloor \mathbf{s} \cdot \mathbf{A}_T(x) + \mathbf{e}_{x_0} \cdot \mathbf{S}_T(x') \rceil_p .$$

Because $\mathbf{e}_{x_0}$ is short, and so is any matrix of the form $\mathbf{G}^{-1}(\cdot)$ (with 0/1 entries), $\mathbf{e}_{x_0} \cdot \mathbf{S}_T(x')$ is short. More precisely, its entries are of magnitude bounded by $\approx (n \log q)^d$, which is much less than q/p because $d \leq e(T)$ and by assumption on q. Therefore, the additive term $\mathbf{e}_{x_0} \cdot \mathbf{S}_T(x')$ is very unlikely the change the final rounded value, i.e., with high probability $F'_{\mathbf{s},\mathbf{e}_0,\mathbf{e}_1}(x) = F_{\mathbf{s}}(x)$ for all the adversary's queries x. Therefore, this hybrid game is statistically indistinguishable from the real attack.

In the next hybrid game, we replace each $\mathbf{s} \cdot \mathbf{A}_b + \mathbf{e}_b$ for $b \in \{0,1\}$ by uniformly random and independent $\mathbf{u}_b$, i.e., the adversary has access to the function

$$F''_{\mathbf{u}_0,\mathbf{u}_1}(x) := \lfloor \mathbf{u}_{x_0} \cdot \mathbf{S}_T(x') \rceil_p = \left\lfloor \mathbf{u}_{x_0} \cdot \mathbf{G}^{-1}(\mathbf{A}_{T_1}(x'_1)) \cdot \underbrace{\prod_{i=2}^{d} \mathbf{G}^{-1}(\mathbf{A}_{T_i}(x'_i))}_{\mathbf{S}'_T(x')} \right\rceil_p . \quad (5.3.2)$$

Since $\mathbf{S}_T(x')$ can be efficiently computed from the public parameters $\mathbf{A}_b$ and the adversary's queries x, this game is computationally indistinguishable from the previous one, under the LWE assumption.

At this point, we would like to be able to proceed by replacing the terms $\mathbf{u}_{x_0} \cdot \mathbf{G}^{-1}(\mathbf{A}_{T_1}(x'_1))$ with some "noisy" variants, then replace those with uniform and independent vectors for all values of $x_0 \| x'_1$, etc. Indeed, this is possible if x'_1 consists of a *single bit* (i.e., if $|T_1| = 1$ and hence $\mathbf{A}_{T_1}(x'_1) = \mathbf{A}_{x'_1}$), using "non-uniform LWE" exactly as is done in [18]. Unfortunately, non-uniform LWE does not appear to be sufficient when x'_1 is more than one bit (i.e., when $|T_1| > 1$), because the matrices $\mathbf{A}_{T_1}(x'_1)$ are not independent for different values of x'_1. And requiring $|T_i| = 1$ for all i makes T have maximal expansion $e(T) = |T| - 1$.

Our main new proof technique is a way to deal with the above issue. Going back to Equation (5.3.2), as "wishful thinking" suppose that each $\mathbf{u}_b$ was of the form $\mathbf{u}_b = \mathbf{s}_b \cdot \mathbf{G}$ for some (uniform, say) $\mathbf{s}_b \in \mathbb{Z}_q^n$. Then the $\mathbf{G}$ factor would undo the decomposition $\mathbf{G}^{-1}(\cdot)$, and the adversary would have access to the function

$$F'''_{\mathbf{s}_0,\mathbf{s}_1}(x) := \lfloor \mathbf{s}_{x_0} \cdot \mathbf{A}_{T_1}(x'_1) \cdot \mathbf{S}'_T(x') \rceil_p = \lfloor \mathbf{s}_{x_0} \cdot \mathbf{A}_{T'}(x') \rceil_p,$$

where T' is the full binary tree obtained from T by removing its leftmost leaf v and promoting v's sibling subtree T_1 to replace their parent. Notice that the above function is just two independent members of our function family instantiated with tree T'. Moreover, T' has expansion $e(T') \leq e(T)$, because expansion is just "left depth." Therefore, the above function would be a PRF simply by induction on $|T|$.

Unfortunately, our "wishful thinking" fails in a very strong sense: a uniformly random $\mathbf{u}$ is highly likely to be *very far* from any vector of the form $\mathbf{s} \cdot \mathbf{G}$. However, because $\mathbb{Z}_q^n \cdot \mathbf{G}$ is a *subgroup* of $\mathbb{Z}_q^{n\ell}$, a uniformly random vector $\mathbf{u} \in \mathbb{Z}_q^{n\ell}$ *can* be decomposed as $\mathbf{u} = \mathbf{s} \cdot \mathbf{G} + \mathbf{v}$ where $\mathbf{s} \in \mathbb{Z}_q^n$ is uniform, and $\mathbf{v}$ is uniform in (some canonical set of representatives of) the quotient group $\mathbb{Z}_q^{n\ell}/(\mathbb{Z}_q^n \cdot \mathbf{G})$ and *independent* of $\mathbf{s}$. Therefore, the function in Equation (5.3.2) is equivalent to the function

$$F'''_{\mathbf{s}_0,\mathbf{s}_1,\mathbf{v}_0,\mathbf{v}_1}(x) := \lfloor \mathbf{s}_{x_0} \cdot \mathbf{A}_{T'}(x') \quad + \quad \mathbf{v}_{x_0} \cdot \mathbf{S}_T(x') \rceil_p,$$

where T' and x' are exactly as in the previous paragraph. Note that $\mathbf{v}_b$ is *not* short, so the extra additive term above does not simply "round away"—but we do not need it to. The main point is that $\mathbf{v}_b$ may be chosen *independently* of (and hence without knowledge of) $\mathbf{s}_b$ by the simulator, and then the additive term may be efficiently computed from it and other public information. Essentially, this allows us to complete the proof by induction on $|T|$. (Again, the actual proof is structured a bit differently, to allow us to simulate the independent additive terms *inside* the rounding operation.)

5.3.2 Proof of Security Theorem

In this section we give the formal proof of Theorem 5.2.3. We restate it here for the reader's convenience.

Theorem 5.2.3. *Let T be any full binary tree, χ be some distribution over $\mathbb{Z}$ that is subgaussian with parameter $r > 0$ (e.g., a bounded or discrete Gaussian distribution with expectation zero), and*

$$q \geq p \cdot r\sqrt{|T|} \cdot (n\ell)^{e(T)} \cdot \lambda^{\omega(1)}.$$

Then over the uniformly random and independent choice of $\mathbf{A}_0, \mathbf{A}_1 \in \mathbb{Z}_q^{n \times n\ell}$, the family $\mathcal{F}_{\mathbf{A}_0,\mathbf{A}_1,T,p}$ with secret key chosen uniformly from $\mathbb{Z}_q^n$ is a secure PRF family, under the decision-$\mathsf{LWE}_{n,q,\chi}$ assumption.

To aid the proof we first define a couple of auxiliary function families. The first family simply consists of the "pre-rounded" counterparts of the functions $F_{\mathbf{s}} \in \mathcal{F} = \mathcal{F}_{\mathbf{A}_0,\mathbf{A}_1,T,p}$.

Definition 5.3.1. *For $\mathbf{A}_0, \mathbf{A}_1 \in \mathbb{Z}_q^{n \times n\ell}$ and a full binary tree T, the family $\mathcal{G} = \mathcal{G}_{\mathbf{A}_0,\mathbf{A}_1,T}$ is the set of functions $G_{\mathbf{s}} \colon \{0,1\}^{|T|} \to \mathbb{Z}_q^{n\ell}$ indexed by some $\mathbf{s} \in \mathbb{Z}_q^n$, and defined as $G_{\mathbf{s}}(x) := \mathbf{s} \cdot \mathbf{A}_T(x)$ (where we define $\mathbf{A}_T(\varepsilon) := \mathbf{G}$ for the empty tree T). We endow $\mathcal{G}$ with the distribution where $\mathbf{s} \leftarrow \mathbb{Z}_q^n$ is chosen uniformly at random.*

Note that $F_{\mathbf{s}}(x) = \lfloor G_{\mathbf{s}}(x) \rceil_p$.

The next family $\tilde{\mathcal{G}}$ consists of functions that are certain "noisy" versions of the functions in $\mathcal{G}$. The family $\mathcal{E}$ of "error functions" used in the definition is a family of functions from $\{0,1\}^{|T|}$ to $\mathbb{Z}^{n\ell}$, and is formally defined in Definition 5.3.5 below. An important point is that the functions in $E \in \mathcal{E}$ have *exponentially* large keys, but they may be efficiently sampled "lazily," as values $E(x)$ are needed. See the discussion following Definition 5.3.5 for details.

Definition 5.3.2. *For $\mathbf{A}_0, \mathbf{A}_1 \in \mathbb{Z}_q^{n \times n\ell}$ and a full binary tree T, the family $\tilde{\mathcal{G}} = \tilde{\mathcal{G}}_{\mathbf{A}_0,\mathbf{A}_1,T}$ is the set of functions $\tilde{G}_{\mathbf{s},E} \colon \{0,1\}^{|T|} \to \mathbb{Z}_q^{n\ell}$ indexed by some $G_{\mathbf{s}} \in \mathcal{G}$ and $E \in \mathcal{E} = \mathcal{E}_{\mathbf{A}_0,\mathbf{A}_1,T}$, and defined as $\tilde{G}_{\mathbf{s},E}(x) := G_{\mathbf{s}}(x) + E(x)$. We endow $\tilde{\mathcal{G}}$ with the distribution where $G_{\mathbf{s}} \leftarrow \mathcal{G}$ and $E \leftarrow \mathcal{E}$ are chosen independently.*

Proof of Theorem 5.2.3. We show below that with overwhelming probability, the rounding of $G_{\mathbf{s}} \in \mathcal{G}$ agrees with the rounding of essentially any corresponding $\tilde{G}_{\mathbf{s},E} \in \tilde{\mathcal{G}}$ on all the attacker's queries, because the outputs of the error functions $E \in \mathcal{E}$ are small. We also prove in Theorem 5.3.7 below that $\tilde{\mathcal{G}}$ is a PRF family *without* any rounding, and hence with rounding as well. It follows that the rounding of $G_{\mathbf{s}} \leftarrow \mathcal{G}$ (i.e., $F_{\mathbf{s}} \leftarrow \mathcal{F}$) cannot be distinguished from a uniformly random function, as desired. We now proceed more formally, by a sequence of games.

Game H_0. This is the real PRF attack game: we choose public parameters $\mathbf{A}_0, \mathbf{A}_1 \leftarrow \mathbb{Z}_q^{n \times n\ell}$ and an $F_{\mathbf{s}} \leftarrow \mathcal{F}$, and the attacker gets $\mathbf{A}_0, \mathbf{A}_1$ and oracle access to $F_{\mathbf{s}}(\cdot)$. Equivalently, we choose $G_{\mathbf{s}} \leftarrow \mathcal{G}$ and the attacker gets oracle access to $\lfloor G_{\mathbf{s}}(\cdot) \rceil_p$.

Game H_1. We choose $\mathbf{A}_0, \mathbf{A}_1$ as above and a $\tilde{G}_{\mathbf{s},E} \leftarrow \tilde{\mathcal{G}}$, by choosing $G_{\mathbf{s}} \leftarrow \mathcal{G}$ and $E \leftarrow \mathcal{E}$. Note that we choose E (and hence $\tilde{G}$) "lazily" as the attacker makes queries (see the remarks following Definition 5.3.5). The attacker is given $\mathbf{A}_0, \mathbf{A}_1$ and oracle access to $\lfloor \tilde{G}_{\mathbf{s},E}(\cdot) \rceil_p$, but with one exception: on query x, define the "bad event" Bad_x for that query to be

$$\left\lfloor \tilde{G}_{\mathbf{s},E}(x) + [-R, R]^{n\ell} \right\rceil_p \neq \{ \lfloor \tilde{G}_{\mathbf{s},E}(x) \rceil_p \},$$

where $R = r\sqrt{|T|} \cdot (n\ell)^{e(T)} \cdot \omega(\sqrt{\log \lambda})$. That is, Bad_x indicates whether any entry of $\tilde{G}_{\mathbf{s},E}(x) \in \mathbb{Z}_q^{n\ell}$ is "too close" to another element of $\mathbb{Z}_q$ that rounds to a different value in $\mathbb{Z}_p$. Note that because $q/p \gg R$, a $y \in \mathbb{Z}_q$ is "too close" in this sense if and only if $\lfloor y - R \rceil_p \neq \lfloor y + R \rceil_p$, so Bad_x can be efficiently detected given only the value of $\tilde{G}_{\mathbf{s},E}(x)$. If Bad_x occurs any of the attacker's queries, then the game immediately aborts.

In Lemma 5.3.6 below, we show that for any fixed $x \in \{0,1\}^{|T|}$, with overwhelming probability over the choice of $E \leftarrow \mathcal{E}$, it is the case that $E(x) \in [-R, R]^{n\ell}$. Therefore, for any fixed x we have $\tilde{G}_{\mathbf{s},E}(x) \in G_{\mathbf{s}}(x) + [-R, R]^{n\ell}$. Hence $\lfloor \tilde{G}_{\mathbf{s},E}(x) \rceil_p = \lfloor G_{\mathbf{s}}(x) \rceil_p$ as long as Bad_x does not occur, and the attacker's queries are answered exactly as they are in H_0, conditioned on the game not aborting. It follows that for any (potentially unbounded) attacker $\mathcal{A}$,

$$\mathbf{Adv}_{H_0,H_1}(\mathcal{A}) \leq \Pr[\text{some } \mathsf{Bad}_x \text{ occurs in } H_1 \text{ with attacker } \mathcal{A}] + \mathrm{negl}(\lambda). \quad (5.3.4)$$

We do not directly bound the probability that some Bad_x occurs in H_1, but instead defer to the analysis of the next game, where we can show that it is indeed negligible.

Game H_2. We choose $\mathbf{A}_0, \mathbf{A}_1$ as above, and U to be a uniformly random function from $\{0,1\}^{|T|}$ to $\mathbb{Z}_q^{n\ell}$ (defined "lazily" as the attacker makes queries). The attacker is given $\mathbf{A}_0, \mathbf{A}_1$ and oracle access to $\lfloor U(\cdot) \rceil_p$, with the same "bad event" and abort condition as in H_1, but defined relative to U instead of $\tilde{G}_{\mathbf{s},E}$.

In Theorem 5.3.7 below, we show that $\tilde{\mathcal{G}}$ is a PRF family under the LWE assumption from the theorem statement, i.e., no efficient adversary can distinguish (given oracle access) between $\tilde{G}_{\mathbf{s},E} \leftarrow \tilde{\mathcal{G}}$ and a uniformly random function $U\colon \{0,1\}^{|T|} \to \mathbb{Z}_q^{n\ell}$. Because the Bad_x event in H_1 (respectively, H_2) for a query x can be tested efficiently given query access to $\tilde{G}$ (resp., U), a trivial simulation implies that for any efficient attacker $\mathcal{A}$, we have $\mathbf{Adv}_{H_1,H_2}(\mathcal{A}) \leq \mathrm{negl}(\lambda)$. For the same reasons, it also follows by a straightforward simulation that for any efficient attacker $\mathcal{A}$,

$$|\Pr[\text{some } \mathsf{Bad}_x \text{ occurs in } H_1 \text{ with } \mathcal{A}] - \Pr[\text{some } \mathsf{Bad}_x \text{ occurs in } H_2 \text{ with } \mathcal{A}]| \leq \mathrm{negl}(\lambda).$$

In H_2, because U is a uniformly random function, for any particular query x the probability that Bad_x occurs is bounded by $(2R+1) \cdot (n\ell) \cdot p/q$, which is $\mathrm{negl}(\lambda)$ by assumption on q. By a union bound over all $\mathrm{poly}(\lambda)$ queries of an efficient $\mathcal{A}$, and then applying Equation (5.3.4), we therefore have that

$$\Pr[\text{some } \mathsf{Bad}_x \text{ occurs in } H_1 \text{ with } \mathcal{A}] = \mathrm{negl}(\lambda) \quad \Rightarrow \quad \mathbf{Adv}_{H_0,H_1}(\mathcal{A}) = \mathrm{negl}(\lambda).$$

Game H_3. We choose $\mathbf{A}_0, \mathbf{A}_1$ and a uniformly random function U as above, and give the attacker oracle access to $\lfloor U(\cdot) \rceil_p$. For each query x we define the event Bad_x as in game H_2, but still answer the query and continue with the game even if Bad_x occurs. From the above analysis of H_2 it follows that for any (potentially unbounded) attacker $\mathcal{A}$ making $\mathrm{poly}(\lambda)$ queries, we have

$$\mathbf{Adv}_{H_2,H_3}(\mathcal{A}) \leq \Pr[\text{some } \mathsf{Bad}_x \text{ occurs in } H_2 \text{ with } \mathcal{A}] = \mathrm{negl}(\lambda).$$

Finally, observe that $\lfloor U(\cdot) \rceil_p$ is a truly random function from $\{0,1\}^{|T|}$ to $\mathbb{Z}_p^{n\ell}$,

up to the bias involved in rounding the uniform distribution on $\mathbb{Z}_q$ to $\mathbb{Z}_p$. Because $q \geq p \cdot \lambda^{\omega(1)}$, this bias is negligible (and there is no bias if p divides q).

By the triangle inequality, it follows that for any efficient $\mathcal{A}$, we have $\mathbf{Adv}_{H_0,H_3}(\mathcal{A}) = \mathrm{negl}(\lambda)$, and this completes the proof. □

We next define the "error function" family $\mathcal{E} = \mathcal{E}_{\mathbf{A}_0,\mathbf{A}_1,T}$ and prove the claims used in the above proof. To define the error functions we first need a couple of simple definitions.

Definition 5.3.3 (Pruning). *For a full binary tree T of at least one node, define its* pruning $T' = pr(T)$ *inductively as follows: if $|T.l| \leq 1$ then $T' := T.r$; otherwise, $T'.l := pr(T.l)$ and $T'.r := T.r$. We let $T^{(i)}$ denote the ith successive pruning of T, i.e., $T^{(0)} = T$ and $T^{(i)} = pr(T^{(i-1)})$.*

In other words, pruning a tree node removes its leftmost leaf v and replaces the subtree rooted at v's parent (if it exists) with the subtree rooted at v's sibling. Notice that pruning cannot increase the tree's expansion (i.e., left depth; see Equation (5.2.4)): $e(T') \leq e(T)$.

Definition 5.3.4. *Given $\mathbf{A}_0, \mathbf{A}_1 \in \mathbb{Z}_q^{n \times n\ell}$ and a full binary tree T of at least one node, define the function $\mathbf{S}_T \colon \{0,1\}^{|T|-1} \to \mathbb{Z}^{n\ell \times n\ell}$ recursively as follows:*

$$\mathbf{S}_T(x) = \begin{cases} \mathbf{I} \text{ (the identity matrix)} & \text{if } |T| = 1 \\ \mathbf{S}_{T.l}(x_l) \cdot \mathbf{G}^{-1}(\mathbf{A}_{T.r}(x_r)) & \text{otherwise,} \end{cases}$$

where $x = x_l \| x_r$ for $|x_l| = |T.l| - 1, |x_r| = |T.r|$.

Notice that if $T' = pr(T)$ and $x = x_1 \| x' \in \{0,1\}^{|T|}$ for $|x_1| = 1$, then it follows directly from the definitions (recalling that $\mathbf{A}_\varepsilon(\varepsilon) = \mathbf{G}$) and by induction that

$$\mathbf{A}_T(x) = \mathbf{A}_{x_1} \cdot \mathbf{S}_T(x'), \tag{5.3.6}$$

$$\mathbf{G} \cdot \mathbf{S}_T(x') = \mathbf{A}_{T'}(x'). \tag{5.3.7}$$

Definition 5.3.5 (Error Functions). *For public matrices $\mathbf{A}_0, \mathbf{A}_1 \in \mathbb{Z}_q^{n \times n\ell}$ and a full binary tree T, the family $\mathcal{E} = \mathcal{E}_{\mathbf{A}_0,\mathbf{A}_1,T}$ consists of functions from $\{0,1\}^{|T|}$ to $\mathbb{Z}^{n\ell}$, defined inductively as follows.*

- *For $|T| = 0$, the sole function in $\mathcal{E}$ is defined simply as $E(\varepsilon) := \mathbf{0}$.*
- *For $|T| \geq 1$, a function in $\mathcal{E}$ is indexed by some $\mathbf{e}_0, \mathbf{e}_1 \in \mathbb{Z}^{n\ell}$ and $E'_0, E'_1 \in \mathcal{E}' = \mathcal{E}_{\mathbf{A}_0,\mathbf{A}_1,T'}$, where T' is the pruning of T. For $x = x_1 \| x' \in \{0,1\}^{|T|}$, the function is defined as*

$$E_{\mathbf{e}_0,\mathbf{e}_1,E'_0,E'_1}(x) := \mathbf{e}_{x_1} \cdot \mathbf{S}_T(x') + E'_{x_1}(x').$$

For a given error function distribution χ over $\mathbb{Z}$, we endow $\mathcal{E}$ with the distribution where $\mathbf{e}_0, \mathbf{e}_1 \leftarrow \chi^{n\ell}$ and $E'_0, E'_1 \leftarrow \mathcal{E}'$ are all chosen independently.

Note that a function $E \in \mathcal{E}$ is fully specified by *exponentially* (in $|T|$) many error vectors (namely, one $\mathbf{e}_w$ for each $w \in \{0,1\}^{\leq |T|}$), and the value $E(x)$ is fully determined by those $\mathbf{e}_w$ where w is a prefix of x (and $\mathbf{A}_0, \mathbf{A}_1$). This large number of error vectors is what prevents $\tilde{\mathcal{G}}$ itself from being usable as a PRF family. However, as needed in the proof of Theorem 5.2.3 (game H_1), a function $E \leftarrow \mathcal{E}$ can be sampled "lazily" as values $E(x)$ are needed, since each value of $E(x)$ depends on only a small number of the error vectors.

We now prove the claim used in the analysis of game H_1 above.

Lemma 5.3.6. Let $\mathbf{A}_0, \mathbf{A}_1 \in \mathbb{Z}_q^{n \times n\ell}$, let T be any full binary tree, and let χ be a subgaussian distribution over $\mathbb{Z}$ with parameter $r > 0$, used to sample the error functions from the family $\mathcal{E} = \mathcal{E}_{\mathbf{A}_0,\mathbf{A}_1,T}$. Then for any $x \in \{0,1\}^{|T|}$ and for $E \leftarrow \mathcal{E}$, the vector $E(x) \in \mathbb{Z}^{n\ell}$ is subgaussian with parameter at most $r\sqrt{|T|} \cdot (n\ell)^{e(T)}$. In particular, for $R = r\sqrt{|T|} \cdot (n\ell)^{e(T)} \cdot \omega(\sqrt{\log \lambda})$ we have $E(x) \in [-R, R]^{n\ell}$ except with negligible probability in λ (over the choice of E).

Proof. First note that for $|T| = 0$ we have $E(x) = \mathbf{0}$, which satisfies the claim. For $|T| \geq 1$ we proceed by induction, assuming the claim for the pruning T' of T, for which $e(T') \leq e(T)$. Let $E = E_{\mathbf{e}_0,\mathbf{e}_1,E'_0,E'_1} \leftarrow \mathcal{E}$. By definition, all the $\mathbf{e}_b \leftarrow \chi^{n\ell}$ and $E'_b \leftarrow \mathcal{E}_{\mathbf{A}_0,\mathbf{A}_1,T'}$ are mutually independent, and $E(x) = \mathbf{e}_{x_1} \cdot \mathbf{S}_T(x') + E'_{x_1}(x')$, where $x = x_1 \| x'$.

- By unrolling the recursion in Definition 5.3.4, we see that $\mathbf{S}_T(x')$ is the product of d separate $\mathbf{G}^{-1}(\cdot)$ components, where $d \leq e(T)$ is the depth of the leftmost leaf of T. The $\mathbf{G}^{-1}(\cdot)$ components in the product are $n\ell$-by-$n\ell$ binary matrices, all of which have spectral norm bounded by $n\ell$, by Lemma 2.1.1. By submultiplicative property of the spectral norm, this implies that $s_1(\mathbf{S}_T(x')) \leq (n\ell)^{e(T)}$.

- Because each vector $\mathbf{e}_{x_1} \leftarrow \chi^{n\ell}$ is subgaussian with parameter r, it follows that $\mathbf{e}_{x_1} \cdot \mathbf{S}_T(x')$ is subgaussian with parameter $r \cdot s_1(\mathbf{S}_T(x')) \leq r \cdot (n\ell)^{e(T)}$.

- By the induction hypothesis, $E'_{x_1}(x')$ is subgaussian with parameter $r\sqrt{|T'|} \cdot (n\ell)^{e(T')} \leq r\sqrt{|T| - 1} \cdot (n\ell)^{e(T)}$, and is independent of $\mathbf{e}_{x_1}$.

- By Pythagorean additivity of subgaussians, we conclude that $E(x)$ is subgaussian with parameter $r\sqrt{|T|} \cdot (n\ell)^{e(T)}$, as claimed.

The "in particular" part of the claim follows from the fact that each entry of $E(x)$ is subgaussian with the same parameter, the Gaussian tail bound for subgaussians, and the union bound. □

We finally prove that the function family $\tilde{\mathcal{G}}$ from Definition 5.3.2 is pseudorandom.

Theorem 5.3.7. *For any $n, q \geq 1$ and error distribution χ over $\mathbb{Z}$, any full binary tree T, and over the uniformly random and independent choice of $\mathbf{A}_0, \mathbf{A}_1 \in \mathbb{Z}_q^{n \times n\ell}$, the family $\tilde{\mathcal{G}} = \tilde{\mathcal{G}}_{\mathbf{A}_0,\mathbf{A}_1,T}$ is pseudorandom, assuming the hardness of decision-$\mathsf{LWE}_{n,q,\chi}$.*

Proof. We proceed through a series of games, one for each bit of the input. In each successive game, we modify the function family $\tilde{\mathcal{G}}$ a little, until we are left with the family of all functions from $\{0,1\}^{|T|}$ to $\mathbb{Z}_q^{n\ell}$ (with uniform distribution), and we show that each successive game is computationally indistinguishable under the LWE assumption from the theorem statement.

To define the games formally, we first need some notation. For a bit string x of length at least i, let $x_{(i)} = x_1 x_2 \cdots x_i$ denote the string of its first i bits, and let $x^{(i)}$ denote the remainder of the string. Where $\mathbf{A}_0, \mathbf{A}_1$ and T are clear from context, let $\mathcal{G}^{(i)} = \mathcal{G}_{\mathbf{A}_0,\mathbf{A}_1,T^{(i)}}$ and similarly for $\mathcal{E}^{(i)}$. Let $\mathcal{P} \subset \mathbb{Z}^{n\ell}$ denote an arbitrary set of representatives of the quotient group $\mathbb{Z}_q^{n\ell}/\mathbb{Z}_q^n \cdot \mathbf{G}$, and define a family of auxiliary functions $\mathcal{V}^{(i)} = \mathcal{V}^{(i)}_{\mathbf{A}_0,\mathbf{A}_1,T}$ as follows.

Definition 5.3.8. *For public matrices $\mathbf{A}_0, \mathbf{A}_1 \in \mathbb{Z}_q^{n\times n\ell}$, a full binary tree T, and $0 \le i \le |T|$, the family $\mathcal{V}^{(i)} = \mathcal{V}^{(i)}_{\mathbf{A}_0,\mathbf{A}_1,T}$ consists of functions from $\{0,1\}^{|T|}$ to $\mathbb{Z}^{n\ell}$, and is defined inductively as follows.*

- *The sole function in $\mathcal{V}^{(0)}$ is defined simply as $V(x) := \mathbf{0}$.*
- *For $i \ge 1$, a function in $\mathcal{V}^{(i)}$ is indexed by some $\mathbf{v}_w \in \mathbb{Z}^{n\ell}$ for every $w \in \{0,1\}^i$, and some $V' \in \mathcal{V}^{(i-1)}$. The function is defined as*

$$V_{\{\mathbf{v}_w\},V'}(x) := \mathbf{v}_{x_{(i)}} \cdot \mathbf{S}_{T^{(i-1)}}(x^{(i)}) + V'(x).$$

  We endow $\mathcal{V}^{(i)}$ with the distribution where the $\mathbf{v}_w \leftarrow \mathcal{P}$ and $V' \leftarrow \mathcal{V}^{(i-1)}$ are all chosen independently and uniformly.

Similarly to the family $\mathcal{E}$ of error functions, the description of a function in $\mathcal{V}^{(i)}$ consists of an exponential (in i) number of $\mathbf{v}_w$ vectors, and can be sampled lazily.

We now define game H_i for $0 \le i \le |T|$.

Game H_i. Choose $\mathbf{A}_0, \mathbf{A}_1 \leftarrow \mathbb{Z}_q^{n \times n\ell}$ independently, and lazily sample $G_{\mathbf{s}_w} \leftarrow \mathcal{G}^{(i)}$ and $E_w \leftarrow \mathcal{E}^{(i)}$ for each $w \in \{0,1\}^i$, and $V \leftarrow \mathcal{V}^{(i)}$. Give the adversary $\mathbf{A}_0, \mathbf{A}_1$ and oracle access to the function

$$H(x) := G_{\mathbf{s}_{x_{(i)}}}(x^{(i)}) + E_{x_{(i)}}(x^{(i)}) + V(x).$$

Claim 5.3.9. Game H_0 corresponds to the real attack game against the family $\tilde{\mathcal{G}}$, and game $H_{|T|}$ corresponds to oracle access to a uniformly random function.

Proof. The first claim follows by definition of $\tilde{\mathcal{G}} = \tilde{\mathcal{G}}_{\mathbf{A}_0,\mathbf{A}_1,T}$, and because $\mathcal{V}^{(0)}$ consists solely of the zero function. For the second claim, for $i = |T|$ we have $x_{(i)} = x$, $x^{(i)} = \varepsilon$, and $T^{(i)} = \varepsilon$ (the empty tree), so by Definitions 5.3.1, 5.3.5, and 5.3.8,

$$H(x) = G_{\mathbf{s}_x}(\varepsilon) + E_x(\varepsilon) + V(x) = \mathbf{s}_x \cdot \mathbf{G} + \mathbf{v}_x + V'(x).$$

Since $\mathbf{s}_x \in \mathbb{Z}_q^n, \mathbf{v}_x \in \mathcal{P}$ are uniformly random and independent for each x, and $\mathcal{P}$ is a set of representatives of the quotient group $\mathbb{Z}_q^{n\ell}/\mathbf{G}^t \cdot \mathbb{Z}_q^n$, the values $\mathbf{s}_x \cdot \mathbf{G} + \mathbf{v}_x \in \mathbb{Z}_q^{n\ell}$ are uniformly random and independent. Since V' is independent of these as well, H is a uniformly random function. □

It remains to prove that successive games are computationally indistinguishable. To do so we define the following games H_i' for $1 \le i \le |T|$.

Game H_i'. Choose $\mathbf{A}_0, \mathbf{A}_1 \leftarrow \mathbb{Z}_q^{n \times n\ell}$ independently, and lazily sample $\mathbf{u}_w \leftarrow \mathbb{Z}_q^{n\ell}$ and $E_w \leftarrow \mathcal{E}^{(i)}$ for each $w \in \{0,1\}^i$, and $V' \leftarrow \mathcal{V}^{(i-1)}$. Give the adversary $\mathbf{A}_0, \mathbf{A}_1$ and oracle access to the function

$$H'(x) = \mathbf{u}_{x_{(i)}} \cdot \mathbf{S}_{T^{(i-1)}}(x^{(i)}) + E_{x_{(i)}}(x^{(i)}) + V(x). \tag{5.3.10}$$

Claim 5.3.10. For $1 \le i \le |T|$, games H_i and H_i' are equivalent.

Proof. We can write each uniformly random $\mathbf{u}_w \in \mathbb{Z}_q^{n\ell}$ for $w \in \{0,1\}^i$ as $\mathbf{u}_w = \mathbf{s}_w \cdot \mathbf{G} + \mathbf{v}_w$, where $\mathbf{s}_w \in \mathbb{Z}_q^n$ and $\mathbf{v}_w \in \mathcal{P}$ are uniformly random and independent. Therefore, we can rewrite the function $H'(\cdot)$ from Equation (5.3.10) as

$$\begin{aligned} H'(x) &= \left(\mathbf{s}_{x_{(i)}} \cdot \mathbf{G} + \mathbf{v}_{x_{(i)}}\right) \cdot \mathbf{S}_{T^{(i-1)}}(x^{(i)}) + E_{x_{(i)}}(x^{(i)}) + V'(x) \\ &= \mathbf{s}_{x_{(i)}} \cdot \mathbf{G} \cdot \mathbf{S}_{T^{(i-1)}}(x^{(i)}) + E_{x_{(i)}}(x^{(i)}) + \left(\mathbf{v}_{x_{(i)}} \cdot \mathbf{S}_{T^{(i-1)}}(x^{(i)}) + V'(x)\right) \\ &= G_{\mathbf{s}_{x_{(i)}}}(x^{(i)}) + E_{x_{(i)}}(x^{(i)}) + V(x), \end{aligned}$$

where in the final equality we have used Equation (5.3.7), and we have defined $V(x)$ to be the second parenthesized component of the previous expression. Notice that all the functions $G_{\mathbf{s}_{x_{(i)}}}$, $E_{x_{(i)}}$, and V are drawn independently from $\mathcal{G}^{(i)}$, $\mathcal{E}^{(i)}$, and $\mathcal{V}^{(i)}$ (respectively), and this proves the claim. □

Claim 5.3.11. For $0 \le i \le |T| - 1$, games H_i and H'_{i+1} are computationally indistinguishable under the LWE assumption from the theorem statement.

Proof. To prove the claim, we design an efficient simulator $\mathcal{S}$ which receives as input a pair of matrices $(\mathbf{A}, \mathbf{B}) \in \mathbb{Z}_q^{n \times 2n\ell} \times \mathbb{Z}_q^{Q \times 2n\ell}$, where $Q = \text{poly}(\lambda)$ is the minimum of 2^i and the number of queries that the adversary makes. The simulator parses $\mathbf{A} = [\mathbf{A}_0 \mid \mathbf{A}_1]$ where $\mathbf{A}_0, \mathbf{A}_1 \in \mathbb{Z}_q^{n \times n\ell}$ and gives them to the adversary. It lazily samples a $V \leftarrow \mathcal{V}^{(i)}$ and an $E_w \leftarrow \mathcal{E}^{(i+1)}$ for every $w \in \{0,1\}^{i+1}$. Then for each query x from the adversary, if a vector $\mathbf{b}_{x_{(i)}}$ has not already been defined, it lets $\mathbf{b}_{x_{(i)}}$ be a previously unused row of $\mathbf{B}$. It parses $\mathbf{b}_{x_{(i)}} = (\mathbf{b}_{x_{(i)}\|0} \mid \mathbf{b}_{x_{(i)}\|1})$, where $\mathbf{b}_{x_{(i)}\|b} \in \mathbb{Z}_q^{n\ell}$ for each $b \in \{0,1\}$. It then answers the query with the value

$$J(x) := \mathbf{b}_{x_{(i+1)}} \cdot \mathbf{S}_{T^{(i)}}(x^{(i+1)}) + E_{x_{(i+1)}}(x^{(i+1)}) + V(x).$$

This completes the description of $\mathcal{S}$.

We now analyze the behavior of $\mathcal{S}$ for the two distributions of $(\mathbf{A}, \mathbf{B})$ from the decision-LWE problem. In both cases, $\mathbf{A}$ is uniformly random and so the public

parameters are properly distributed. When $\mathbf{B}$ is uniformly random, it can be seen by inspection that the function J is drawn from the same distribution as the function H' in game H'_{i+1} described in Equation (5.3.10), so the simulator exactly emulates game H'_{i+1}.

We now analyze the other case, namely, $\mathbf{B} = \mathbf{S} \cdot \mathbf{A} + \mathbf{E}$ for independent $\mathbf{S} \leftarrow U(\mathbb{Z}_q^{Q \times n})$ and $\mathbf{E} \leftarrow \chi^{Q \times 2n\ell}$. Then letting $\mathbf{s}_{x_{(i)}}, (\mathbf{e}_{x_{(i)}\|0} \mid \mathbf{e}_{x_{(i)}\|1})$ respectively be the rows of $\mathbf{S}, \mathbf{E}$ corresponding to the row of $\mathbf{B}$ used as $\mathbf{b}_{x_{(i)}}$, we have

$$\begin{aligned} J(x) &= \left(\mathbf{s}_{x_{(i)}} \cdot \mathbf{A}_{x_{i+1}} + \mathbf{e}_{x_{(i)}\|x_{i+1}}\right) \cdot \mathbf{S}_{T^{(i)}}(x^{(i+1)}) + E_{x_{(i+1)}}(x^{(i+1)}) + V(x) \\ &= \mathbf{s}_{x_{(i)}} \cdot \mathbf{A}_{T^{(i)}}(x^{(i)}) + \left(\mathbf{e}_{x_{(i)}\|x_{i+1}} \cdot \mathbf{S}_{T^{(i)}}(x^{(i+1)}) + E_{x_{(i)}\|x_{i+1}}(x^{(i+1)})\right) + V(x) \\ &= G_{\mathbf{s}_{x_{(i)}}}(x^{(i)}) + E_{x_{(i)}}(x^{(i)}) + V(x), \end{aligned}$$

where in the second equality we have used Equation (5.3.6), and in the last expression we have defined $E_{x_{(i)}}(x^{(i)})$ to be the parenthesized component from the previous expression. Notice that by the distributions of all the variables, the functions $G_{\mathbf{s}_w}$, E_w (for each queried prefix $w \in \{0,1\}^i$) and V are all drawn independently from $\mathcal{G}^{(i)}$, $\mathcal{E}^{(i)}$, and $\mathcal{V}^{(i)}$, so in this case the simulator exactly emulates game H_i.

Because the two LWE input distributions are computationally indistinguishable by assumption and $\mathcal{S}$ is efficient, it follows that H_i and H'_{i+1} are computationally indistinguishable, and the claim is proved. □

By repeated application of Claims 5.3.10 and 5.3.11, we have that $H_0 \overset{c}{\approx} H'_1 \equiv H_1 \overset{c}{\approx} H'_2 \equiv \cdots \equiv H_{|T|-1} \overset{c}{\approx} H'_{|T|} \equiv H_{|T|}$, and so $H_0 \overset{c}{\approx} H_{|T|}$ by the triangle inequality. This completes the proof of Theorem 5.3.7. □

5.4 Related Work

This construction is reminiscent of those from several works on fully homomorphic encryption, attribute-based encryption, and garbled circuits, e.g., [36, 26, 17]. In particular, these works obtain relatively good LWE assumptions and parameters by

appropriately scheduling "bit decomposition" operations to ensure small noise growth, usually at the expense of increased sequentiality. This construction also falls within this theme, though our proof techniques are completely different.

CHAPTER VI

CONSTRAINED KEY-HOMOMORPHIC PSEUDORANDOM FUNCTIONS

6.1 Overview

Constrained PRFs. A *constrained* PRF for a family of sets $\mathcal{S} \subseteq \mathcal{P}(\mathcal{X})$ has the property that, given any key k and set $S \in \mathcal{S}$, one can efficiently compute a "constrained" key k_S that enables evaluation of $F_k(x)$ on all inputs $x \in S$, while the values $F_k(x)$ for $x \notin S$ remain pseudorandom even given k_S.

Constrained PRFs were introduced in three independent works [20, 49, 21]. All three papers note that the classical GGM construction [39] already gives a prefix-constrained PRF, where from a key $k \in \{0,1\}^n$, for any $w \in \{0,1\}^{\leq n}$ one can compute a key k_w that enables the computation of $F_k(x)$ for all inputs x that start with w. This construction is an instance of a prefix-fixing constrained PRF. Boneh and Waters [20] construct bit-fixing and circuit-fixing constrained PRFs from multilinear maps. In the bit-fixing construction, for every $w \in \{0,1,?\}$ one can compute a key k_w that enables the computation of $F_k(x)$ for any x for which $x_i = w_i$ when $w_i \neq ?$. The more general circuit-constrained construction allows generating constrained keys for any circuit C, where with k_C one can evaluate the PRF on input x if and only if $C(x) = 1$.

Prefix-constrained PRFs (or rather, "punctured" PRFs, which can be constructed from them) are a main tool in almost all the applications of indistinguishability obfuscation [33, 88, 79]. The papers [20, 49, 21] discuss many more applications of constrained PRFs.

Key-Homomorphic Constrained PRFs. In this chapter, we construct PRFs that are simultaneously constrained *and* key homomorphic, where the homomorphic property holds even for constrained keys. We show that the LWE-based key-homomorphic PRFs from Chapter 5 are essentially already *prefix-constrained* PRFs, using a (non-obvious) definition of constrained keys and associated group operation. Moreover, the constrained keys themselves are pseudorandom, and the constraining and evaluation functions can all be computed in low depth. The latter feature can be important for applications of obfuscation, e.g., [33, 88, 79], where the use of low-depth constrained/punctured PRFs may avoid the need for costly "bootstrapping" operations and fully homomorphic encryption.

6.1.1 Low-Depth Prefix-Fixing PRFs from LWE

In Section 6.3 we construct key-homomorphic prefix-fixing constrained PRFs from the LWE assumption, and hence from the conjectured hardness of worst-case lattice problems [86, 80, 23]. In addition, natural instantiations of this construction have polylogarithmic circuit depth. To our knowledge, these are the first sublinear-depth constrained PRFs (whether key-homomorphic or not), and as such they can admit much more efficient obfuscation under existing paradigms. (Recall that the basic GGM construction, which yields a prefix-constrained PRF, is highly sequential.)

Our LWE-based construction is an extension of the key-homomorphic PRFs of Chapter 5. We show that this construction can be made prefix-constrained, and that the constraining algorithm is also key-homomorphic. Notably, the approximation factors for the underlying LWE assumption are essentially the same as before, e.g., they can be as small as quasi-polynomial $\lambda^{\omega(1)}$ in the security parameter.

To show all this, we start with the observation that the security proof of the noisy function in Chapter 5 is very "GGM-like," i.e., it proceeds in a sequence of hybrids, one for each successive bit of the PRF input. However, the functions computed in the

hybrids do not quite fit the usual GGM paradigm, because each successive output of the PRG is broken into two pieces: one piece is fed as input into the next PRG step, while the "leftover" piece is retained and then later "folded" back into the final output of all the PRG steps. A natural way to define a constrained key for a *partial* function evaluation, then, is to include all the leftover pieces in the constrained key—and this is indeed the approach we take.

The main technical challenge we face is in defining a suitable *group structure* on the leftover pieces, for key homomorphism. At first sight, this appears easy: since the leftover pieces are eventually combined with the final PRG output via a linear function, it would appear that one could simply add constrained keys by adding their corresponding leftover pieces. While this does indeed work—at least syntactically—it yields a useless construction! The problem is that essentially any application of key-homomorphic (constrained) PRFs will require a statistical "secret sharing"-like property on the (constrained) secret keys. For example, the sum of any fixed key with a uniformly random key must be uniformly random, so that the original key is completely hidden. Formally speaking, for any particular constraint we need the space of constrained keys to be a *finite* additive group (so that it supports a uniform distribution), and for the function to be key-homomorphic under this group structure.

Resolving the difficulty. Going back to Construction 5.2.2, the leftover pieces in constrained keys come from a certain finite subset $\mathcal{P} \subset \mathbb{Z}^m$, namely, a fundamental region of a special lattice $\mathcal{L}$. Obviously, the sum of two uniformly random $\mathcal{P}$-elements is *not* uniform in $\mathcal{P}$—indeed, it is typically not in $\mathcal{P}$ at all! So we cannot naïvely use the ambient group $\mathbb{Z}^m$ (which is infinite). Another idea would be to use the *finite* quotient group $\mathbb{Z}^m/\mathcal{L}$, i.e., addition modulo the lattice. This also does not work, because the function is not key-homomorphic under this form of addition.

Our solution to the above problem involves a novel method of adding modulo $\mathcal{L}$

"with carries." That is, the sum of two leftover $\mathcal{P}$-elements is mapped back to $\mathcal{P}$ by reducing modulo the lattice $\mathcal{L}$, i.e., shifting by an appropriate lattice vector $\mathbf{x} \in \mathcal{L}$. The vector $\mathbf{x}$ is then treated as a "carry" term that is "folded into" the sum of the next two $\mathcal{P}$-elements in the key, and so on. (The ultimate effect is analogous to grade-school addition, except that here the "base" in which the "numbers" are written is a high-dimensional lattice.) We show that by appropriately defining the effect of the carry terms, the PRF is indeed key-homomorphic under this form of addition.

All of the above applies to the so-called "noisy" version of the construction, an intermediate object that has perfect constraining, homomorphic, and pseudorandomness properties, but high circuit depth and (even worse) *exponentially* large keys. Similar to Chapters 4 and 5, we show that by appropriately "rounding" this noisy construction, the keys and depth can be made small while preserving the other desirable properties (at least against computationally bounded attackers). Interestingly, this rounding transformation requires us to work with a "geometrically nice" set $\mathcal{P}$ of representatives modulo the special lattice $\mathcal{L}$ (which fortunately exists), whereas Chapter 5 works with any set of representatives.

6.2 Key-Homomorphic Constrained Pseudorandom Functions

We now formally define key-homomorphic constrained pseudorandom functions. We model constrainability as a directed acyclic graph (DAG) on some (typically huge) set of nodes. We restrict our attention to DAGs having a unique node that has no incoming edges, called the *root node*.

Definition 6.2.1. *A* constrained function family $\mathcal{C}$ *is given by:*

- *a directed acyclic graph* $D = (V, E)$ *with unique root node* $r \in V$*,*
- *for each node* $u \in V$*, a* key space $\mathcal{K}_u$ *with an efficiently samplable probability distribution* $\mathcal{D}_u$ *over it, and a space of* auxilliary inputs $\mathcal{V}_u$*,*

- *for every edge* $(u, w) \in E$*, an efficiently computable* constraining function $C_{u,w} \colon \mathcal{V}_u \times \mathcal{K}_u \to \mathcal{V}_w \times \mathcal{K}_w$.

The functions $C_{u,w}$ *must satisfy the following* consistency *property: for any nodes* $u, w \in V$ *and any two paths* $P = (u = u_0, u_1, \ldots, u_k = w)$ *and* $P' = (u = u'_0, u'_1, \ldots, u'_\ell = w)$ *from* u *to* w*, we have that*

$$C_{u_{k-1},u_k} \circ \cdots \circ C_{u_1,u_2} \circ C_{u_0,u_1} = C_{u'_{\ell-1},u'_\ell} \circ \cdots \circ C_{u'_1,u'_2} \circ C_{u'_0,u'_1}.$$

For notational convenience, we let $C_{u,w} \colon \mathcal{V}_u \times \mathcal{K}_u \to \mathcal{V}_w \times \mathcal{K}_w$ *denote the above (composed) functions, and also define* $C_u := C_{r,u}$ *for any node* $u \in V$ *that is reachable from the root node* r*. For consistency with the typical PRF notation, we define* $F_{v,k}(u) = C_u(v, k)$ *(and to also cover constrained PRFs, if* u *represents a subset of inputs then* $\mathsf{Constrain}_{v,k}(u) = C_u(v, k)$*) for* $v \in \mathcal{V}_v$ *and* $k \in \mathcal{K}_u$*.*

Lastly, a constrained function family may also have a Setup *algorithm, which accepts the security parameter* λ *in unary, samples some (public) parameters that are provided as input to all of the other algorithms and sets up the domains* $\{\mathcal{K}_u\}, \{\mathcal{V}_u\}$*.*

For the reader who may be familiar with constrained PRFs, we stress that in the above definition, the DAG nodes roughly correspond with (subsets of) *PRF inputs*, while the input key, auxiliary key pairs (v_u, k_u) for a node u correspond to *constrained keys*. Despite these rough correspondences, we stress that in our model there are no distinct notions of PRF "inputs" or "outputs," only DAG nodes. This is without loss of generality: a PRF input can simply be represented as a node w with no outgoing edges, and the corresponding output is the pair (v_w, k_w), or even simply the key k_w. In fact, our model is somewhat more general because it allows for defining and proving the pseudorandomness of *constrained keys* themselves (even for nodes having outgoing edges), which can be useful in certain settings.

Definition 6.2.2. *Pseudorandomness for a constrained function family* $\mathcal{C} = (D = (V, E), \{\mathcal{K}_u\}, \{\mathcal{V}_u\}, \{C_{u,v}\})$ *is defined as follows. It is parameterized by a subset* $R \subseteq V$ *of what we call "challenge" nodes. We consider two closely related experiments ("games"), called "real" and "ideal," which proceed as follows:*

1. Initialize: *If the family has a* Setup *algorithm, it is run and its output is provided to the adversary, who in turn provides an auxiliary value* $v_r \in \mathcal{V}_r$ *for the root node* $r \in V$*. The key value for the root* r*,* $k_r \leftarrow \mathcal{K}_r$*, is chosen according to the associated distribution* $\mathcal{D}_r$*.*

2. Query: *The adversary adaptively issues queries* $u \in V$*, subject to the condition that no query in* R *and any other query have a common descendant in* D*. That is, there are no distinct queries* $u \in R$*,* $u' \in V$ *and node* $w \in V$ *such that there exists a (possibly trivial) path from* u *to* w *and one from* u' *to* w*.*

   - *In the "real game," every query* u *is answered with* $(v_u, k_u) = F_{v_r,k_r}(u) = C_u(v_r, k_r)$*.*
   - *In the "ideal game," the auxiliary output* v_u *is answered as in the real game. For the output key, if* $u \in V \setminus R$ *then it is also answered as in the real game, otherwise it is answered with an independent value* $k_u^* \leftarrow \mathcal{D}_u$*. (Repeated queries are answered consistently.)*

The family $\mathcal{C}$ *is said to be* pseudorandom *if for any polynomial-time adversary, its advantage in distinguishing the real and ideal games is negligible in the security parameter.*

In short, the definition above means that constrained keys *for the set* R *of challenge nodes* are pseudorandom. The condition on legal queries is necessary to prevent trivial distinguishers that work by observing the inconsistency of the ideal-game answers. In a bit more detail, given answers $(v_u, k_u), (v_{u'}, k_{u'})$ for some nodes $u \in R$, $u' \in V$

(respectively) that have a common descendant $w \in V$, the distinguisher could check whether $C_{u,w}(v_u, k_u) = C_{u',w}(v_{u'}, k_{u'})$. This always holds in the real game, but in the ideal game, where k_u is chosen independently of everything else, it would typically fail to hold.

Definition 6.2.3. *A constrained function family is* (key) homomorphic *if all the key and auxiliary spaces together $\mathcal{V}_u \times \mathcal{K}_u$ are additive groups and if the constraining functions $C_{u,w}$ are additive homomorphisms, i.e., for every $(u, w) \in E$ and every $(v_1, k_1), (v_2, k_2) \in \mathcal{V}_u \times \mathcal{K}_u$, we have*

$$C_{u,w}(v_1, k_1) + C_{u,v}(v_2, k_2) = C_{u,w}((v_1, k_1) + (v_2, k_2)) \ .$$

For key-homomorphic PRFs, all applications we know of implicitly require the spaces $\mathcal{V}_u \times \mathcal{K}_u$ to be *finite* groups, and the associated distributions $\mathcal{D}_u$ to be *uniform* distributions. In short, this is because the security proofs all rely on statistical "secret sharing"-type properties, e.g., the sum of any group element and a uniformly random one is uniformly random. Our final construction has finite key spaces with uniform distributions.

6.3 Prefix-Fixing Construction from LWE

In this section we prove that variants of the LWE-based key-homomorphic PRF of Chapter 5 also support prefix constraints, and that the constraining functions are key-homomorphic as well. After recalling some standard background and notation in Section 6.3.1, the contents of this section have the following high-level structure:

- In Section 6.3.2 we define a key-homomorphic, prefix-constrained pseudorandom function family called Constrain, which we refer to as the "noisy" family. However, the functions in this family are highly sequential, with circuit depth proportional to the input length. More significantly, they have huge keys, of

size *exponential* in the input length, so they cannot actually be used in reality. The purpose of defining this family is to give us a baseline object that has "perfect" consistency, homomorphic, and pseudorandomness properties (but terrible space and depth complexity), which we rely upon in the later subsections.

- In Section 6.3.3 we specialize the noisy $\mathsf{Constrain}$ family to be "errorless," i.e., all error terms are set to zero. We call the resulting family $\mathsf{PConstrain}$. As a specialization of $\mathsf{Constrain}$, it inherits that latter's perfect consistency and homomorphic properties. We show that the $\mathsf{PConstrain}$ functions (1) have small keys, (2) can be computed in low depth (e.g., logarithmic in the input length) by a slight modification to the $\mathsf{Constrain}$ algorithms, and (3) have outputs that are "close" to those of the noisy $\mathsf{Constrain}$ functions (under a mild condition on the input). However, we are still not quite done yet, because the errorless $\mathsf{PConstrain}$ functions are not pseudorandom.

- In Section 6.3.4 we combine the previous two families to obtain our final family $\overline{\mathsf{PConstrain}}$ that has essentially all the desired properties: small keys, low depth, pseudorandomness, consistency, and almost-homomorphism. (The latter two of these properties do not hold perfectly, but only *computationally*: under the LWE assumption, no efficient adversary can detect a violation of either property.) The $\overline{\mathsf{PConstrain}}$ functions are defined simply as appropriately *rounded* functions from the *errorless* family $\mathsf{PConstrain}$. As such, they inherit the latter's small keys and low depth. In addition, they are pseudorandom because they coincide with rounded versions of the *noisy* (and pseudorandom) $\mathsf{Constrain}$ functions. This correspondence follows from the fact that the (unrounded) errorless $\mathsf{PConstrain}$ and noisy $\mathsf{Constrain}$ functions are "close," and the rounding precision is taken to be sufficiently coarse to conceal this difference. Finally, consistency and almost-homomorphism hold computationally for $\overline{\mathsf{PConstrain}}$ essentially because

rounding can be seen as adding a particular kind of (deterministic) error, so $\overline{\mathsf{PConstrain}}$ may be seen as an instantiation of $\mathsf{Constrain}$.

6.3.1 Preliminaries

In this chapter for convenience we always let the modulus $q = 2^\ell \geq 2$ be a power of two. We recall the gadget matrix $\mathbf{G}$ and the bit-decomposition operation $\mathbf{G}^{-1}(\cdot)$ from Section 2.4. We let $\mathcal{P} \subseteq \mathbb{Z}^{n\ell}$ denote a certain set of canonical representatives of the additive quotient group $\mathbb{Z}_q^{n\ell}/(\mathbb{Z}_q^n \cdot \mathbf{G})$. Specifically, as shown in [70], we can use[1]

$$\mathcal{P} := \{-\tfrac{q}{4}, \ldots, \tfrac{q}{4} - 1\}^{n\ell} \ .$$

We define a bijection $\mathsf{Decode}\colon \mathbb{Z}_q^{n\ell} \to \mathcal{P} \times \mathbb{Z}_q^n$ as $\mathsf{Decode}(\mathbf{u}) = (\mathbf{v}, \mathbf{s})$, where

$$\mathbf{u} = \mathbf{v} + \mathbf{s} \cdot \mathbf{G} \ .$$

As shown in [70], there is an efficient algorithm for computing Decode in depth proportional to $\ell = \log q$, and clearly $\mathsf{Decode}^{-1}(\mathbf{v}, \mathbf{s}) = \mathbf{v} + \mathbf{s} \cdot \mathbf{G}$.

Binary trees. A *full* binary tree T is one in which each node is either a leaf, or has two (nonempty) children. We let $|T|$ denote the number of leaves in T, and index the leaves from 0 to $|T| - 1$ by the inorder traversal of T. If $|T| \geq 1$, we let $T.l$ and $T.r$ respectively denote its left and right subtrees, both of which are nonempty.

Given matrices $\mathbf{A}_0, \mathbf{A}_1 \in \mathbb{Z}_q^{n \times n\ell}$, we define the function $\mathbf{A}_T(x)\colon \{0,1\}^{|T|} \to \mathbb{Z}_q^{n \times n\ell}$, similarly to Chapter 5, as follows:

$$\mathbf{A}_T(x) := \begin{cases} \mathbf{A}_x & \text{if } |T| = 1, \\ \mathbf{A}_{T.l}(x_l) \cdot \mathbf{G}^{-1}(\mathbf{A}_{T.r}(x_r)) & \text{otherwise,} \end{cases}$$

[1] This choice of $\mathcal{P}$ is possible because we have taken q to be a power of two. It may be possible to generalize our results to other values of q using the alternative lattice bases given in [70], but it seems to substantially complicate the proofs.

where in the second case we parse the input $x = x_l x_r$ where $|x_l| = |T.l|$ and $|x_r| = |T.r|$.

Rounding. For a positive integer e, we define the function $\lfloor \cdot \rfloor_e \colon \mathbb{Z} \to e\mathbb{Z}$ as $\lfloor x \rfloor_e := \lfloor x/e \rfloor \cdot e$, i.e., it rounds x down to the nearest integer multiple of e.[2] We also extend $\lfloor \cdot \rfloor_e$ component-wise to vectors and matrices.

6.3.2 "Noisy" Function Family $\mathcal{C}$

As in the previous chapters, we first define and analyze a certain family $\mathcal{C}$ of "noisy" constraining functions, which have *huge* (exponential-size) keys, because each key contains many error terms. To avoid technical complications related to efficient computation on exponential-size inputs, throughout this section the error terms are always sampled "lazily," i.e., not until they are needed. In Section 6.3.2.1 we show that the constraining functions are consistent, in Section 6.3.2.2 we show that they are homomorphisms under an appropriate group operation on the key spaces, and in Section 6.3.2.3 we show that the family is pseudorandom under an appropriate LWE assumption.

The public parameters of the noisy family are two matrices $\mathbf{A}_0, \mathbf{A}_1 \in \mathbb{Z}_q^{n \times n\ell}$, chosen uniformly at random. Following Definitions 6.2.1 and 6.2.3, to describe our family we need to give a DAG with a unique root node, a key space with an additive group structure for each node in the DAG, and a constraining function for each edge in the DAG.

DAG. For a full binary tree T, our DAG D corresponds to prefix-fixing constraints on $\{0,1\}^{|T|}$, i.e., the nodes are identified with the strings in $\{0,1\}^{\leq |T|}$, and there is

[2] We point out that this function differs slightly from the "modular" rounding function considered in the previous chapters, which mapped $\mathbb{Z}_q$ to $\mathbb{Z}_p$ as $\lfloor x \rceil_p = \lfloor x \cdot p/q \rceil \bmod p$. Here e corresponds with q/p, but the rounding input and output have the same "scale." Also, for technical reasons the floor functions is more natural for our needs here.

an edge (w, wx) for every w and $x \neq \varepsilon$ such that $|wx| \leq |T|$. This DAG clearly has a unique root node, namely, the empty string ε.

Key Space and Distribution. For any full binary tree T and $0 \leq j \leq |T|$, we define,

$$\mathcal{E}_{T,j} := \prod_{y \in \{0,1\}^{\leq |T|-j}} \mathbb{Z}^{n\ell} = (\mathbb{Z}^{n\ell})^{2^{|T|-j+1}-1} \ .$$

In $\mathcal{E}_{T,j}$, the several $\mathbb{Z}^{n\ell}$-components (which represent error vectors) are indexed by the binary strings of length at most $|T| - j$, which is why there are $2^{|T|-j+1} - 1$ components. We define a distribution $\mathcal{W}_{T,j}$ on $\mathcal{E}_{T,j}$ as follows

$$\mathcal{W}_{T,j} := (\chi^{nl})^{2^{|T|-j+1}-1} \ ,$$

where χ is some error distribution over $\mathbb{Z}$ that will be used in the security proofs later.

For each $w \in \{0,1\}^{\leq |T|}$, the key space $\mathcal{K}_{T,w}$ and associated distribution are defined as:

$$\mathcal{K}_{T,w} := \mathbb{Z}_q^n \times \mathcal{E}_{T,|w|} \ ,$$
$$\mathcal{D}_{T,w} := U(\mathbb{Z}_q^n) \times \mathcal{W}_{T,|w|} \ .$$

Note that the set $\mathcal{K}_{T,w}$ does not depend on the string w itself, only its length.

Auxiliary Space. We define the auxiliary key spaces $\mathcal{V}_{T,w}$, for $w \in \{0,1\}^{\leq |T|}$ as

$$\mathcal{V}_{T,w} := \begin{cases} \{\varepsilon\} & \text{if } |w| = 0, \\ \mathcal{V}_{T.l,w} & \text{if } 0 < |w| < |T.l|, \\ \mathcal{P} \times \mathcal{V}_{T.r,w_r} & \text{if } |T.l| \leq |w| < |T|, \\ \mathcal{P} & \text{if } |w| = |T|, \end{cases} \tag{6.3.5}$$

where we parse $w = w_l w_r$ for $|w_l| = |T.l|$ in the third case above. As in the definition of the key space, we see that $\mathcal{V}_{T,w}$ depends only on the length $|w|$ of w. In words: for

$0 \leq |w| < |T|$, $\mathcal{V}_{T,w}$ has one $\mathcal{P}$-component for each left-child subtree "hanging off" the path from the root to the $|w|$th leaf. (Recall that we index the leaves starting from zero.) In what follows, we often refer to the subtree associated with a particular $\mathcal{P}$-component of an element of $\mathcal{V}_{T,w}$. The final case is defined so that $\mathcal{V}_{T.l,w} = \mathcal{V}_{T,w} = \mathcal{P}$ for $|w| = |T.l|$, which is a convenient identity for our recursive algorithms.

To make $\mathcal{V}_{T,w} \times \mathcal{K}_{T,w}$ an additive group (for $|w| > 0$), we stress that we do *not* simply treat it as a product group of its components—indeed, $\mathcal{P} \subset \mathbb{Z}^{n\ell}$ is not even closed under addition, so it is not a group. Instead, in Section 6.3.2.2 below we define a special addition operation on $\mathcal{V}_{T,w} \times \mathbb{Z}_q^n$ to make it a group. Then $\mathcal{V}_{T,w} \times \mathcal{K}_{T,w}$ is simply the product group of this group with $\mathcal{E}_{T,|w|}$, with the usual addition operation on the latter.

Constraining functions. It remains to define (consistent) constraining functions

$$\mathsf{Constrain}_{T,w,x} \colon \mathcal{V}_{T,w} \times \mathcal{K}_{T,w} \to \mathcal{V}_{T,wx} \times \mathcal{K}_{T,wx}$$

for all strings w, x such that $x \neq \varepsilon$ and $|wx| \leq |T|$; for convenience, we also define $\mathsf{Constrain}_{T,w,\varepsilon}$ to be the identity function. Functional pseudocode for the constraining functions is given in Algorithm 1. Note that for clarity and convenience in the analysis later on, Algorithm 1 actually defines the "helper" functions $\mathsf{Constrain}'_{T,w,x} \colon \mathcal{V}_{T,w} \times \mathcal{K}_{T,w} \to \mathcal{V}_{T,wx} \times \mathbb{Z}_q^n$, which take errors as *input*, but do not *output* any errors. The full constraining functions are then defined simply as

$$\mathsf{Constrain}_{T,w,x}(\mathbf{v}, \mathbf{s}, (\mathbf{e}_y)_{|y| \leq |T|-|w|}) := (\mathsf{Constrain}'_{T,w,x}(\mathbf{v}, \mathbf{s}, (\mathbf{e}_y)) \,,\, (\mathbf{e}_{xy})_{|y| \leq |T|-|wx|}) \ ,$$

for $\mathbf{v} \in \mathcal{V}_{T,w}$ and $\mathbf{s} \in \mathbb{Z}_q^n$.

The constraining functions are defined recursively on the tree structure. In the base case $|T| = 1$, for key $(\mathbf{s}, (\mathbf{e}_x)_{x \in \{0,1\}^{\leq 1}}) \in \mathcal{V}_{T,\varepsilon} \times \mathcal{K}_{T,\varepsilon} = \mathbb{Z}_q^n \times (\mathbb{Z}^{n\ell})^3$, we simply compute and decode the "noisy" value $\mathbf{s}\mathbf{A}_x + \mathbf{e}_x \in \mathbb{Z}_q^{n\ell}$. There are three recursive cases,

depending on whether we are constraining entirely within the left subtree, within the right subtree, or across the two subtrees. In the first two cases, we simply recurse on the appropriate subtree. In the third case, we recursively constrain over the remainder of the left subtree, then over the desired portion of the right subtree. Lastly, whenever we finish constraining over an *entire* (sub)tree we need to appropriately "fold" the results—which consist of some value in $\mathcal{P} \subset \mathbb{Z}^{n\ell}$ from the left subtree, and some value in $\mathcal{P} \times \mathbb{Z}_q^n$ from the completed right subtree—into a value in $\mathcal{P} \times \mathbb{Z}_q^n$ for the entire tree.

We remark that it would have been sufficient to define functions $\mathsf{Constrain}_{T,x,w}$ for $|x| = 1$ alone. Indeed, by Lemma 6.3.1 below it follows that our pseudocode is actually *equivalent* to the sequential composition of such functions, and hence has circuit depth proportional to $|x|$. We choose to present the constraining functions for general x because in Section 6.3.4 we show that a slight modification yields highly parallel functions. We also point out that although our presentation is (necessarily) quite different, our constraining functions correspond to the *partial* evaluations of the noisy function family from Chapter 5, which the simulator computes internally when answering queries in the security proof.

6.3.2.1 Consistency

Lemma 6.3.1 (Consistency). For any full binary tree T, parameters $\mathbf{A}_0, \mathbf{A}_1$, and strings w, x, z where $|wxz| \leq |T|$, we have that

$$\mathsf{Constrain}_{T,wx,z} \circ \mathsf{Constrain}_{T,w,x} = \mathsf{Constrain}_{T,w,xz} \ .$$

Proof. We proceed by induction on $|T|$. The base case of $|T| = 1$ is trivial, because $\mathsf{Constrain}_{T,w,\varepsilon}$ is the identity function.

We have three inductive cases. In the first two cases, where $|wxz| < |T.l|$ or $|T.l| \leq |w|$, the claim follows immediately by the inductive hypothesis on $T.l$ or $T.r$,

Algorithm 1 $\mathsf{Constrain}'_{T,w,x}\colon \mathcal{V}_{T,w} \times \mathcal{K}_{T,w} \to \mathcal{V}_{T,wx} \times \mathbb{Z}_q^n$ for $|wx| \le |T|$, $x \neq \varepsilon$

Require: $\mathbf{k} = (\mathbf{v}, \mathbf{s}, (\mathbf{e}_y)_{|y| \le |T|-|w|}) \in \mathcal{V}_{T,w} \times \mathbb{Z}_q^n \times \mathcal{E}_{T,|w|}$

if $|T| = 1$ **then** ▷ Base case.
  return $\mathsf{Decode}(\mathbf{s} \cdot \mathbf{A}_x + \mathbf{e}_x)$
else if $|wx| < |T.l|$ **then** ▷ Entirely in left subtree
  return $\mathsf{Constrain}'_{T.l,w,x}\big(\mathbf{v}, \mathbf{s}, (\mathbf{e}_y)_{|y| \le |T.l|-|w|}\big)$ ▷ . . . so just recurse.
else if $|T.l| \le |w|$ **then** ▷ Entirely in right subtree
  parse $w = w_l w_r$ where $|w_l| = |T.l|$ and $\mathbf{v} = (\mathbf{v}_l, \mathbf{v}_r) \in \mathcal{P} \times \mathcal{V}_{T.r,w_r}$
  let $(\mathbf{w}_r, \mathbf{s}_r) = \mathsf{Constrain}'_{T.r,w_r,x}\big(\mathbf{v}_r, \mathbf{s}, (\mathbf{e}_y)_{|y| \le |T|-|w|}\big)$ ▷ . . . so recurse.
  if $|wx| < |T|$ **then** ▷ Doesn't complete the tree. . .
    return $((\mathbf{v}_l, \mathbf{w}_r), \mathbf{s}_r)$ ▷ . . . so append results.
  else ▷ Completes tree (so $\mathbf{w}_r \in \mathcal{P}$). . .
    return $\mathsf{Decode}(\mathbf{v}_l \cdot \mathbf{G}^{-1}(\mathbf{A}_{T.r}(w_r x)) + \mathsf{Decode}^{-1}(\mathbf{w}_r, \mathbf{s}_r))$ ▷ . . . so fold results.
  end if
else ▷ Constrains across both subtrees ($|w| < |T.l| \le |wx|$) . . .
  parse $x = x_l x_r$ where $|wx_l| = |T.l|$
  let $(\mathbf{w}_l, \mathbf{s}_l) = \mathsf{Constrain}'_{T.l,w,x_l}\big(\mathbf{v}, \mathbf{s}, (\mathbf{e}_y)_{|y| \le |T.l|-|w|}\big)$ ▷ . . . complete left subtree
  return $\mathsf{Constrain}'_{T,wx_l,x_r}\big(\mathbf{w}_l, \mathbf{s}_l, (\mathbf{e}_{x_l y})_{|y| \le |T.r|}\big)$ ▷ . . . then self-recurse to finish.
end if

respectively. The last inductive case is $|w| \le |T.l| < |wxz|$. We analyze this in two subcases.

The first subcase is $|w| \le |T.l| \le |wx|$. Here we parse $x = x_l x_r$ with $|wx_l| = |T.l|$. By definition,

$$\mathsf{Constrain}_{T,w,x} = \mathsf{Constrain}_{T,wx_l,x_r} \circ \mathsf{Constrain}_{T.l,w,x_l},$$

$$\mathsf{Constrain}_{T,w,xz} = \mathsf{Constrain}_{T,wx_l,x_r z} \circ \mathsf{Constrain}_{T.l,w,x_l}.$$

The claim then follows by composing $\mathsf{Constrain}_{T,wx,z}$ on the left of both sides of the first equation above, and invoking the second case above (because $|T.l| \le |wx_l|$).

The second subcase is $|wx| \le |T.l|$. This proceeds essentially identically to the first subcase, where we instead parse $z = z_l z_r$ with $|wxz_l| = |T.l|$. □

6.3.2.2 Homomorphism

Before we can prove that the constraining functions are homomorphisms, we must make $\mathcal{V}_{T,w} \times \mathcal{K}_{T,w}$ an additive group. We do so by defining a special group operation

$\oplus$ on the set $\mathcal{V}_{T,w} \times \mathbb{Z}_q^n$. We then treat $\mathcal{V}_{T,w} \times \mathcal{K}_{T,w} = \mathcal{V}_{T,w} \times \mathbb{Z}_q^n \times \mathcal{E}_{T,|w|}$ as the product group of $\mathcal{V}_{T,w} \times \mathbb{Z}_q^n$, under $\oplus$, with $\mathcal{E}_{T,|w|}$, under its usual addition operation. For convenience, we overload $\oplus$ to also refer to the group operation for this product group, where the intended domain should be clear by context.

To aid the recursive definitions, we define $\oplus_{\mathbf{t}}$ to take an auxiliary input $\mathbf{t} \in \mathbb{Z}_q^n$, which should be thought of as a kind of "carry" term that comes from reducing the sum of two $\mathcal{P}$-elements (in $\mathbb{Z}^{n\ell}$) back to $\mathcal{P}$. Initializing this carry input to zero yields the group operation, and in fact, for simplicity of exposition, we often suppress the $\mathbb{Z}_q^n$ carry term if it is equal to zero. As with any group operation, we remark that the group $\mathcal{V}_{T,w} \times \mathbb{Z}_q^n$ of the operands is implicit from the context. Note that unlike the definitions of the domains themselves, this one depends on the bit string w. We define $\oplus_{\mathbf{t}}$ to mirror the definition of $\mathcal{V}_{T,w}$ (Equation (6.3.5)).

$$(\mathbf{v}_1, \mathbf{s}_1) \oplus_{\mathbf{t}} (\mathbf{v}_2, \mathbf{s}_2) :=$$

$$\begin{cases} \mathbf{t} + \mathbf{s}_1 + \mathbf{s}_2 & \text{if } |w| = 0, \\ (\mathbf{v}_1, \mathbf{s}_1) \oplus_{\mathbf{t}} (\mathbf{v}_2, \mathbf{s}_2) & \text{if } 0 < |w| < |T.l|, \\ \Big(\overline{\mathbf{v}}_l, \big((\mathbf{v}_{1,r}, \mathbf{s}_1) \oplus_{\overline{\mathbf{t}}} (\mathbf{v}_{2,r}, \mathbf{s}_2)\big)\Big) & \text{if } |T.l| \le |w| < |T|, \\ \mathsf{Decode}\big(\mathbf{t} \cdot \mathbf{A}_T(w) + \mathsf{Decode}^{-1}(\mathbf{v}_1, \mathbf{s}_1) + \mathsf{Decode}^{-1}(\mathbf{v}_2, \mathbf{s}_2)\big) & \text{if } |w| = |T|, \end{cases}$$

where in the second case we recurse into $T.l$, interpreting the operands to be in $\mathcal{V}_{T.l,w} \times \mathbb{Z}_q^n$, and in the third case, we parse $w = w_l w_r$ for $|w_l| = |T.l|$ and $\mathbf{v}_1 = (\mathbf{v}_{1,l}, \mathbf{v}_{1,r}) \in \mathcal{P} \times \mathcal{V}_{T.r,w_r}$, and similarly for $\mathbf{v}_2$ in the third case, and let $(\overline{\mathbf{v}}_l, \overline{\mathbf{t}}) = \mathsf{Decode}(\mathbf{t} \cdot \mathbf{A}_{T.l}(w_l) + \mathbf{v}_{1,l} + \mathbf{v}_{2,l})$. It can immediately be seen by definition that $\oplus_{\mathbf{t}}$ is commutative. It also follows by a straightforward proof by induction over $|w|$ that $\oplus$ is associative, which we state below.

Fact 6.3.2 (Associativity). For any full binary tree T, bit-string $w \in \{0,1\}^{\le |T|}$, $\mathbf{k}_1, \mathbf{k}_2, \mathbf{k}_3 \in$

$\mathcal{V}_{T,w} \times \mathbb{Z}_q^n$ and $\mathbf{t}_1, \mathbf{t}_2 \in \mathbb{Z}_q^n$, the following is true:

$$\mathbf{k}_1 \oplus_{\mathbf{t}_1} (\mathbf{k}_2 \oplus_{\mathbf{t}_2} \mathbf{k}_3) = (\mathbf{k}_1 \oplus_{\mathbf{t}_1} \mathbf{k}_2) \oplus_{\mathbf{t}_2} \mathbf{k}_3 = \mathbf{k}_1 \oplus_{\mathbf{t}_1+\mathbf{t}_2} (\mathbf{k}_2 \oplus_{\mathbf{0}} \mathbf{k}_3) \ .$$

We now prove that the $\mathsf{Constrain}$ functions are homomorphisms.

Lemma 6.3.3 (Homomorphism). For any parameters $\mathbf{A}_0, \mathbf{A}_1$ and any full binary tree T, any bit strings w, x such that $|wx| \leq |T|$, and any $\mathbf{t} \in \mathbb{Z}_q^n$ and $\mathbf{k}, \mathbf{k}' \in \mathcal{V}_{T,w} \times \mathcal{K}_{T,w}$, we have

$$\mathsf{Constrain}_{T,w,x}(\mathbf{k} \oplus_{\mathbf{t}} \mathbf{k}') = \mathsf{Constrain}_{T,w,x}(\mathbf{k}) \oplus_{\mathbf{t}} \mathsf{Constrain}_{T,w,x}(\mathbf{k}') \ . \tag{6.3.10}$$

In particular, by setting $\mathbf{t} = \mathbf{0}$ we have that $\mathsf{Constrain}_{T,w,x}$ is an additive homomorphism.

Proof. Because $\mathsf{Constrain}$ simply passes an appropriate subset of the input $\mathbb{Z}^{n\ell}$-components (the error terms) to the output, and by the product group structure of $\mathcal{K}_{T,w}$, it suffices to prove the claim for $\mathsf{Constrain}'_{T,w,x}$. The lemma is trivially true for $x = \varepsilon$, so from now on we assume that $x \neq \varepsilon$. Let $\overline{\mathbf{k}} = \mathbf{k} \oplus_{\mathbf{t}} \mathbf{k}'$.

As usual, we proceed by induction over $|T|$. In the base case, where $w = \epsilon$ and $|x| = |T| = 1$, parse $\mathbf{k} = (\mathbf{s}, (\mathbf{e}_y))$, $\mathbf{k}' = (\mathbf{s}', (\mathbf{e}'_y))$, $\overline{\mathbf{k}} = (\overline{\mathbf{s}}, (\overline{\mathbf{e}}_y))$. Then $\overline{\mathbf{s}} = \mathbf{t} + \mathbf{s} + \mathbf{s}'$ and $\overline{\mathbf{e}}_x = \mathbf{e}_x + \mathbf{e}'_x$, so

$$\begin{aligned}
\mathsf{Constrain}'_{T,\epsilon,x}(\overline{\mathbf{s}}, (\overline{\mathbf{e}}_y)) &= \mathsf{Decode}(\overline{\mathbf{s}} \cdot \mathbf{A}_x + \overline{\mathbf{e}}_x) \\
&= \mathsf{Decode}((\mathbf{t} + \mathbf{s} + \mathbf{s}') \cdot \mathbf{A}_x + (\mathbf{e}_x + \mathbf{e}'_x)) \\
&= \mathsf{Decode}(\mathbf{t} \cdot \mathbf{A}_x + (\mathbf{s} \cdot \mathbf{A}_x + \mathbf{e}_x) + (\mathbf{s}' \cdot \mathbf{A}_x + \mathbf{e}'_x)) \\
&= \mathsf{Constrain}'_{T,w,x}(\mathbf{s}, (\mathbf{e}_y)) \oplus_{\mathbf{t}} \mathsf{Constrain}'_{T,w,x}(\mathbf{s}', (\mathbf{e}'_y)) \ .
\end{aligned}$$

We now consider the inductive cases, where we parse $\mathbf{k} = (\mathbf{v}, \mathbf{s}, (\mathbf{e}_y))$ and similarly for $\mathbf{k}'$ and $\overline{\mathbf{k}}$. The first inductive case, where $|wx| < |T.l|$, follows immediately by

the definitions of $\mathsf{Constrain}'_{T,w,x}$ and $\oplus$ and the inductive hypothesis for $T.l$. For the second inductive case, where $|T.l| \leq |w| < |T|$, parse $w = w_l w_r$ for $|w_l| = |T.l|$ and $\mathbf{v} = (\mathbf{v}_l, \mathbf{v}_r), \mathbf{v}' = (\mathbf{v}'_l, \mathbf{v}'_r), \overline{\mathbf{v}} = (\overline{\mathbf{v}}_l, \overline{\mathbf{v}}_r) \in \mathcal{P} \times \mathcal{V}_{T.r,w_r}$, and note that by definition,

$$(\overline{\mathbf{v}}_r, \overline{\mathbf{s}}) = (\mathbf{v}_r, \mathbf{s}) \oplus_{\overline{\mathbf{t}}} (\mathbf{v}'_r, \mathbf{s}'), \tag{6.3.11}$$

$$\overline{\mathbf{v}}_l + \overline{\mathbf{t}} \cdot \mathbf{G} = \mathbf{t} \cdot \mathbf{A}_{T.l}(w_l) + \mathbf{v}_l + \mathbf{v}'_l \tag{6.3.12}$$

for some $\overline{\mathbf{t}} \in \mathbb{Z}_q^n$. As in the code for $\mathsf{Constrain}'$, let $(\mathbf{w}_r, \mathbf{s}_r) = \mathsf{Constrain}'_{T.r,w_r,x}(\mathbf{v}_r, \mathbf{s})$ (for ease of exposition, we suppress the error terms $\mathbf{e}_y$ in this and what follows) and similarly for $(\mathbf{w}'_r, \mathbf{s}'_r), (\overline{\mathbf{w}}_r, \overline{\mathbf{s}}_r)$. Then by the inductive hypothesis for $T.r$ and Equation (6.3.11), we have

$$\mathsf{Constrain}'_{T.r,w_r,x}(\overline{\mathbf{v}}_r, \overline{\mathbf{s}}) = (\mathbf{w}_r, \mathbf{s}_r) \oplus_{\overline{\mathbf{t}}} (\mathbf{w}'_r, \mathbf{s}'_r) \ . \tag{6.3.14}$$

Now if $|wx| < |T|$, then by definition of $\mathsf{Constrain}'_{T,w,x}$ and $\oplus$ (respectively), the left- and right-hand sides of Equation (6.3.10) are respectively just $\overline{\mathbf{v}}_l$ prepended to the left- and right-hand sides of Equation (6.3.14), so they are equal. But if $|wx| = |T|$ (that is, if $\mathcal{V}_{T,wx} = \mathcal{P}$), we proceed by applying Decode^{-1} to both sides of Equation (6.3.14), add $\overline{\mathbf{v}}_l \cdot \mathbf{G}^{-1}(\mathbf{A}_{T.r}(w_r x))$, and apply Decode. For the left-hand side we get exactly $\mathsf{Constrain}'_{T,w,x}(\overline{\mathbf{k}})$, which is the left-hand side of Equation (6.3.10). On the right-hand side, by definition of $\oplus$ (over $T.r$), by Equation (6.3.12), and by definition of $\mathbf{A}_T(\cdot)$, we get Decode applied to

$$\begin{aligned}
&\overline{\mathbf{v}}_l \cdot \mathbf{G}^{-1}(\mathbf{A}_{T.r}(w_r x)) + \mathsf{Decode}^{-1}((\mathbf{w}_r, \mathbf{s}_r) \oplus_{\overline{\mathbf{t}}} (\mathbf{w}'_r, \mathbf{s}'_r))) \\
&\quad = (\overline{\mathbf{v}}_l + \overline{\mathbf{t}} \cdot \mathbf{G}) \cdot \mathbf{G}^{-1}(\mathbf{A}_{T.r}(w_r x)) + \mathsf{Decode}^{-1}(\mathbf{w}_r, \mathbf{s}_r) + \mathsf{Decode}^{-1}(\mathbf{w}'_r, \mathbf{s}'_r) \\
&\quad = (\mathbf{t} \cdot \mathbf{A}_{T.l}(w_l) + \mathbf{v}_l + \mathbf{v}'_l) \cdot \mathbf{G}^{-1}(\cdots) + \mathsf{Decode}^{-1}(\mathbf{w}_r, \mathbf{s}_r) + \mathsf{Decode}^{-1}(\mathbf{w}'_r, \mathbf{s}'_r) \\
&\quad = \mathbf{t} \cdot \mathbf{A}_T(wx) + (\mathbf{v}_l \cdot \mathbf{G}^{-1}(\cdots) + \mathsf{Decode}^{-1}(\mathbf{w}_r, \mathbf{s}_r)) + (\mathbf{v}'_l \cdot \mathbf{G}^{-1}(\cdots) \\
&\qquad + \mathsf{Decode}^{-1}(\mathbf{w}'_r, \mathbf{s}'_r)) \ .
\end{aligned}$$

As desired, this is the right-hand side of Equation (6.3.10), by definition of $\mathsf{Constrain}'_{T,w,x}$ and $\oplus$.

For the third inductive case, where $|w| < |T.l| \leq |wx|$, the claim follows by writing $\mathsf{Constrain}'_{T,w,x} = \mathsf{Constrain}'_{T,wx_l,x_r} \circ \mathsf{Constrain}'_{T,w,x_l}$ where $x = x_l x_r$ for $|wx_l| = |T.l|$, then applying the inductive hypothesis for $T.l$ and $T.r$. This completes the proof of Lemma 6.3.3. □

6.3.2.3 Pseudorandomness

We now show that the function family $\mathcal{C}$ defined above is pseudorandom according to Definition 6.2.2. This follows from the PRG-like property demonstrated in Lemma 6.3.4 below, and Theorem 6.3.6, which is in the style of proof in prior works [20, 49, 21] that show that the GGM construction [39] instantiated with such a PRG family yields a prefix-constrained PRF.

Lemma 6.3.4. Let T be any full binary tree, $w \in \{0,1\}^{|T|-1}$ be any string and let $\mathbf{v}$ be any auxiliary input provided by the adversary. Then assuming the hardness of decision-$\mathsf{LWE}_{n,q,\chi}$, for $(\mathbf{s}, (\mathbf{e}_y)) \leftarrow \mathcal{D}_{T,w}$ we have

$$\Big(\mathsf{Constrain}'_{T,w,0}(\mathbf{v}, \mathbf{s}, (\mathbf{e}_y)), \mathsf{Constrain}'_{T,w,1}(\mathbf{v}, \mathbf{s}, (\mathbf{e}_y))\Big) \overset{c}{\approx} U\Big((\mathcal{P} \times \mathbb{Z}_q^n) \times (\mathcal{P} \times \mathbb{Z}_q^n)\Big) \ .$$

We now make a note about the assumption that $w \in \{0,1\}^{|T|-1}$. In any evaluation of $\mathsf{Constrain}'_{T,w,b}$ by Algorithm 1, for an arbitrary $w \in \{0,1\}^{<|T|}$ and $b \in \{0,1\}$, after dealing with all the inductive cases, we reach a point where we are completing some subtree. If we are recursing in the right subtree without completing it, then we "peel off" a $\mathcal{P}$-component from the $\mathcal{V}_{T,w}$ part which we prepend to the results. We cannot make any pseudorandomness claim for this part of the output that is essentially just "passed through" the evaluation of $\mathsf{Constrain}'_{T,w,b}$. Thus, in this lemma, we prove that $\mathsf{Constrain}'_{T,w,b}$ is *almost* a PRG in the sense that whatever *fresh* outputs are generated by the function are distributed uniformly. However, as we will see later, since the auxiliary inputs are public (and in fact can be chosen adversarially) and we do not

make a claim about their pseudorandomness in Definition 6.2.2, we only need that the key output (that is, the $\mathbb{Z}_q^n$ component) be distributed uniformly.

Proof of Lemma 6.3.4. We first observe that for $(\mathbf{s}, (\mathbf{e}_y)_{y\in\{0,1\}^{|T|-|w|}}) \in \mathcal{K}_{T,w} = \mathbb{Z}_q^n \times \mathcal{E}_{T,|w|}$ distributed according to $\mathcal{D}_{T,w}$, all the components are independent, and the values $\mathsf{Constrain}_{T,w,b}(\mathbf{v}, \mathbf{s}, (\mathbf{e}_y))$ for $b = 0, 1$ can be efficiently computed given only $\mathbf{v}$, $\mathbf{b}_b = \mathbf{s}\cdot\mathbf{A}_b + \mathbf{e}_b$. This is done simply by adapting the base case (Line 1) of Algorithm 1 to use $\mathbf{b}_b$ instead of $\mathbf{s}, \mathbf{e}_b$.

To prove the lemma, we design an efficient simulator $\mathcal{S}$ whose input is a pair $(\mathbf{A}, \mathbf{b}) \in \mathbb{Z}_q^{n\times 2n\ell} \times \mathbb{Z}_q^{2n\ell}$. The simulator parses $\mathbf{A} = [\mathbf{A}_0 \mid \mathbf{A}_1]$ where $\mathbf{A}_0, \mathbf{A}_1 \in \mathbb{Z}_q^{n\times n\ell}$ and releases them as the public parameters, and also parses $\mathbf{b} = (\mathbf{b}_0, \mathbf{b}_1)$ for $\mathbf{b}_0, \mathbf{b}_1 \in \mathbb{Z}_q^{n\ell}$. It then uses $\mathbf{v} \in \mathcal{V}_{T,w}$ to output the (purported) values of $\mathsf{Constrain}_{T,w,b}$ as described above.

We now analyze the output for the two distributions of $(\mathbf{A}, \mathbf{b})$, LWE or uniform. In either case, $\mathbf{A}$ is distributed uniformly at random in the appropriate domain, so the public parameters are properly distributed. When $\mathbf{b}$ is distributed according to the LWE distribution, that is, $\mathbf{b} = \mathbf{s}\cdot\mathbf{A} + \mathbf{e}$ for independent $\mathbf{s} \leftarrow \mathbb{Z}_q^n$ and $\mathbf{e} \leftarrow \chi^{2n\ell}$, as argued above, the outputs are distributed according to $(\mathsf{Constrain}'_{T,w,0}(\mathbf{v}, \mathbf{s}, (\mathbf{e}_y)), \mathsf{Constrain}'_{T,w,1}(\mathbf{v}, \mathbf{s}, (\mathbf{e}_y)))$ for an appropriately distributed $(\mathbf{s}, (\mathbf{e}_y)) \leftarrow \mathcal{D}_{T,w}$ (where $(\mathbf{e}_0, \mathbf{e}_1) = \mathbf{e}$).

Finally, in the case that $\mathbf{b}$ is distributed uniformly, we prove that the outputs $\big((\mathbf{w}_0, \mathbf{t}_0), (\mathbf{w}_1, \mathbf{t}_1)\big) \in (\mathcal{P} \times \mathbb{Z}_q^n)^2$ are uniform and independent by induction on the tree T. By assumption on w, that is $|w| = |T| - 1$, we only deal with two inductive cases. The base case, where $|T| = 1$ and $w = \varepsilon$, is easily seen to hold by inspection. In the remaining case, $|w| = |T| - 1 \geq |T.l|$, where $\mathsf{Constrain}$ recurses on the right subtree and we parse $\mathbf{v} = (\mathbf{v}_l, \mathbf{v}_r)$ for $\mathbf{v}_l \in \mathcal{P}$. By the induction hypothesis on $T.r$, the $\mathsf{Constrain}'_{T.r,w_r,b}(\mathbf{v}_r, \mathbf{s}, (\mathbf{e}_y))$ outputs for $b \in \{0, 1\}$ are uniform and independent in $\mathcal{P} \times \mathbb{Z}_q^n$, and therefore the Decode^{-1} of these values are is uniform and independent

in $\mathbb{Z}_q^{n\ell}$, and since the $\mathbf{v}_l$ term is fixed, the folded and decoded outputs are distributed uniformly and independently in $\mathcal{P} \times \mathbb{Z}_q^n$, as required.

Because the two distributions for $(\mathbf{A}, \mathbf{b})$ are computationally indistinguishable by assumption, and $\mathcal{S}$ is efficient by construction, the lemma follows. □

This leads to an immediate corollary which we state below and which finds use later.

Corollary 6.3.5. Let T be a full binary tree and let $w \in \{0,1\}^{\leq |T|}$. For $(\mathbf{s}, (\mathbf{e}_y)) \leftarrow \mathcal{D}_{T,\epsilon}$,

$$\mathsf{Constrain}'_{T,\epsilon,w}(\mathbf{s}, (\mathbf{e}_y)) \overset{c}{\approx} U(\mathcal{V}_{T,w} \times \mathbb{Z}_q^n)$$

under the $\mathsf{LWE}_{n,q,\chi}$ assumption.

Proof. As usual, the proof is by induction over $|T|$. The statement is trivially true if $w = \epsilon$. The case when $|T| = |w|$ easily follows by Lemma 6.3.4. The case when $|w| < |T.l|$ follows from the induction hypothesis. In the remaining case, when $|T.l| \leq |w| < |T|$, we parse $w = w_l w_r$ for $|w_l| = |T.l|$ and note that $(\mathbf{w}, \mathbf{t}) = \mathsf{Constrain}'_{T.l,\epsilon,w_l}(\mathbf{s}, (\mathbf{e}_y)_{|y| \leq |T.l|}) \overset{c}{\approx} U(\mathcal{P} \times \mathbb{Z}_q^n)$ by the discussion above (since $|w_l| = |T.l|$). We then have that $\mathbf{k}_r = \mathsf{Constrain}'_{T.r,\epsilon,w_r}(\mathbf{t}, (\mathbf{e}_{w_l y})_{|y| \leq |T.r|}) \overset{c}{\approx} U(\mathcal{V}_{T.r,w_r} \times \mathbb{Z}_q^n)$ by induction hypothesis on $T.r$. The final output is then $(\mathbf{w}, \mathbf{k}_r) \overset{c}{\approx} U(\mathcal{V}_{T,w} \times \mathbb{Z}_q^n)$, as required.□

We now prove the pseudorandomness of the function family $\mathcal{C}$.

Theorem 6.3.6 (Pseudorandomness). *The construction described in Section 6.3.2 is pseudorandom according to Definition 6.2.2 for the set of challenge nodes* $R = \{0,1\}^{\leq |T|}$ *under the* $\mathsf{LWE}_{n,q,\chi}$ *assumption.*

Proof. Firstly, note that since $R = \{0,1\}^{\leq |T|}$, *every* query made by the adversary would be on a challenge node. Secondly, note that since the number of components in $\mathcal{E}_{T,0}$ is exponential in $|T|$, we sample lazily from $\mathcal{W}_{T,0}$ in all the hybrid games ahead. That is, $\mathcal{E}_{T,0}$ components are sampled as they are needed in the computations. (This

ensures all algorithms do not trivially run in exponential time just to enumerate all such components, most of which are never used.)

We prove this theorem by going through a series of $Q+1$ hybrids, H_0 to H_Q, where $Q = \text{poly}(\lambda)$ is the number of queries made by the adversary. In hybrid H_i, for every $0 \le i \le Q$, the root is initialized with a $(\mathbf{s}_\epsilon, (\mathbf{e}_y)) \leftarrow \mathcal{D}_{T,\epsilon}$ (note that (1) there is no auxiliary associated with the root and (2) the error vectors are sampled lazily, as they are needed), the first i queries are answered according to the ideal game and the last $Q-i$ queries are answered according to the real game in Definition 6.2.2. We note that the game H_0 corresponds to the real game and H_Q corresponds to the ideal game, and thus, if we can show that $H_i \overset{c}{\approx} H_{i+1}$ for every $0 \le i < Q$ then the theorem follows by a triangle inequality over the $Q = \text{poly}(\lambda)$ hybrids.

Now, to show that indeed $H_i \overset{c}{\approx} H_{i+1}$, we define a series of $|T|+1$ hybrids $H_{i,0}$ through $H_{i,|T|}$. For any bit-string w, we use the notation $w_{i,j}$ for $1 \le i \le j \le |w|$ to denote the substring $w_i \cdots w_j$. In particular, $w_{1,i}$ denotes the i bit prefix of w. For completeness, we define $w_{i,j} = \varepsilon$ for $i > j$. In each hybrid $H_{i,j}$, for $1 \le j \le |T|$, the first i queries are answered exactly as in H_i, that is, according to the ideal game.

Now, when we receive the $(i+1)$th query w^*, we let $j^* = \min(|w^*|, j)$ and sample $\mathbf{s}_{w^*_{1,k-1}\overline{w^*_k}}$ for $1 \le k \le j^*$ and $\mathbf{s}_{w^*_{1,j^*}}$, all uniformly from $\mathbb{Z}_q^n$. We also store $\mathbf{v}_{w^*_{1,k-1}\overline{w^*_k}}$ for $1 \le k \le j^*$ and $\mathbf{v}_{w^*_{1,j^*}}$ as follows: each such $\mathbf{v}_w$ is the $\mathcal{V}_{T,w}$ component of $\mathsf{Constrain}'_{T,\epsilon,w}(\mathbf{s}_\epsilon, (\mathbf{e}_y))$. We finally answer this query with the output of $\mathsf{Constrain}'_{T,w^*_{1,j^*},w^*_{j^*+1,|w|}}(\mathbf{v}_{w^*_{1,j^*}}, \mathbf{s}_{w^*_{1,j^*}}, (\mathbf{e}_y))$.

For the final $Q-(i+1)$ queries, we now detail how to answer any legal query $w \in \{0,1\}^{\le T}$ in hybrid $H_{i,j}$. Since w cannot share descendants with w^* there has to be some index $1 \le k \le \min(|w|, |w^*|)$ such that $w_k = \overline{w^*_k}$. There are then two cases, depending on whether $k \le j^*$.

- If $k \le j^*$, then we output $\mathsf{Constrain}_{T,w^*_{1,k-1}\overline{w^*_k},w_{k+1,|w|}}(\mathbf{v}_{w^*_{1,k-1}\overline{w^*_k}}, \mathbf{s}_{w^*_{1,k-1}\overline{w^*_k}}, (\mathbf{e}_y))$.
- If $k > j^*$, then we output $\mathsf{Constrain}_{T,w^*_{1,j^*},w_{j^*+1,|w|}}(\mathbf{v}_{w^*_{1,j^*}}, \mathbf{s}_{w^*_{1,j^*}}, (\mathbf{e}_y))$.

Note that $H_{i,0} \equiv H_i$.

We next prove that for $0 \leq j < |T|$, $H_{i,j} \overset{c}{\approx} H_{i,j+1}$ under the LWE assumption in the hypothesis. We note that the only queries that could change in going from $H_{i,j}$ to $H_{i,j+1}$ are ones which have $w^*_{1,j}$ as a prefix. In particular if $|w^*| \geq j$ (so that $j^* = j$), then a query of the form $w = w^*_{1,j^*}\overline{w^*_{j^*+1}}w_{j+2,|w|}$ is answered as

- $\mathsf{Constrain}_{T,w^*_{1,j^*},\overline{w^*_{j^*+1}}w_{j^*+2,|w|}}(\mathbf{v}_{w^*_{1,j^*}}, \mathbf{s}_{w^*_{1,j^*}}, (\mathbf{e}_y))$ in $H_{i,j}$, and
- $\mathsf{Constrain}_{T,w^*_{1,j^*}\overline{w^*_{j^*+1}},w_{j^*+2,|w|}}(\mathbf{v}_{w^*_{1,j^*}\overline{w^*_{j^*+1}}}), \mathbf{s}_{w^*_{1,j^*}\overline{w^*_{j^*+1}}}, (\mathbf{e}_y))$ in $H_{i,j+1}$.

By consistency (Lemma 6.3.1), we have

$$\mathsf{Constrain}_{T,w^*_{1,j^*},\overline{w^*_{j^*+1}}w_{j^*+2,|w|}} = \mathsf{Constrain}_{T,w^*_{1,j^*}\overline{w}^*_{j^*+1},w_{j^*+2,|w|}} \circ \mathsf{Constrain}_{T,w^*_{1,j^*},\overline{w}^*_{j^*+1}} \ .$$

A similar sequence of equations holds for queries of the form $w = w^*_{1,j^*+1}w_{j^*+2,|w|}$. Thus, it remains to show that

$$\begin{aligned}\Big(\mathsf{Constrain}'_{T,w^*_{1,j^*},w^*_{j^*+1}}(\mathbf{v}_{w^*_{1,j^*}}, \mathbf{s}_{w^*_{1,j^*}}, (\mathbf{e}_y)), \mathsf{Constrain}'_{T,w^*_{1,j^*},\overline{w}^*_{j^*+1}}(\mathbf{v}_{w^*_{1,j^*}}, \mathbf{s}_{w^*_{1,j^*}}, (\mathbf{e}_y))\Big) \\ \overset{c}{\approx} \Big((\mathbf{v}_{w^*_{1,j^*+1}}, \mathbf{s}_{w^*_{1,j^*+1}}), (\mathbf{v}_{w^*_{1,j^*}\overline{w}^*_{j^*+1}}, \mathbf{s}_{w^*_{1,j^*}\overline{w}^*_{j^*+1}})\Big) \qquad (6.3.17)\end{aligned}$$

(because the rest of the errors on both sides are chosen from $\mathcal{W}_{T,j^*+1}$, and thus distributed identically). Now, we know that the part of $\mathbf{v}_{w^*_{1,j^*}}$ that is passed-through in the computation of $\mathsf{Constrain}'_{T,w^*_{1,j^*},b}$ (for $b \in \{0,1\}$) is distributed exactly as the corresponding components of $\mathbf{v}_{w^*_{1,j^*+1}}$ and $\mathbf{v}_{w^*_{1,j^*}\overline{w}^*_{j^*+1}}$, because they are both results of evaluations of $\mathsf{Constrain}'_{T,\epsilon,w^*_{1,j^*}b}(\mathbf{s}_\epsilon, (\mathbf{e}_y))$. Now, under the LWE assumption from the hypothesis, Lemma 6.3.4 holds, and thus we know that the rest of the $\mathcal{P} \times \mathbb{Z}_q^n$ output of $\mathsf{Constrain}'_{T,w^*_{1,j^*},b}$ is computationally indistinguishable from uniform for $b = w^*_{j^*+1}, \overline{w}^*_{j^*+1}$. To establish the claim that Equation (6.3.17) holds, we just need the $\mathbb{Z}_q^n$ component to be uniform, which we have.

Note that the $(i+1)$-th query w^* in game $H_{i,|T|}$ is answered according to the ideal game (since in this game $j^* = |w^*|$), that is the output key is distributed according

to $\mathcal{D}_{T,w^*}$. However, the subsequent queries are not quite answered according to the real game. We now go through another series of hybrids $H'_{i,j}$ for $|T| \geq j \geq 0$, which in effect "reverse" their $H_{i,j}$ counterparts. Hybrid $H'_{i,j}$ is identical to $H_{i,j}$, except that the $(i+1)$-th query w^* is now answered as in the ideal game, that is, the key output is $(\mathbf{v}_{w^*}, \mathbf{s}_{w^*}, (\mathbf{e}_y)) \leftarrow \mathcal{D}_{T,w^*}$. Firstly, note that $H_{i,|T|} \equiv H'_{i,|T|}$. Next, note that $H'_{i,j+1} \overset{c}{\approx} H'_{i,j}$ for every $j^* > j \geq 0$ by exactly the same argument that proves $H_{i,j} \overset{c}{\approx} H_{i,j+1}$ above. Lastly, note that $H'_{i,0} \equiv H_{i+1}$. Summarizing, we have

$$H_i \equiv H_{i,0} \overset{c}{\approx} \ldots \overset{c}{\approx} H_{i,|T|} \equiv H'_{i,|T|} \overset{c}{\approx} \ldots \overset{c}{\approx} H'_{i,0} \equiv H_{i+1} \ ,$$

and this completes the proof. □

6.3.3 Parallel Errorless Function Family

In this subsection we consider the "errorless" variants of our Constrain functions, which we call PConstrain, and show that they can be computed in low depth. We also show that the output of PConstrain is typically close to that of Constrain, when the errors used in the latter are small.

6.3.3.1 Defining PConstrain

The PConstrain functions simply correspond to the Constrain functions with all the error vectors set to zero, i.e., $\mathsf{PConstrain}_{T,w,x}(\mathbf{v}, \mathbf{s}) = \mathsf{Constrain}'_{T,w,x}(\mathbf{v}, \mathbf{s}, \mathbf{0})$. In particular, this implies that the PConstrain functions are both consistent and homomorphisms, because the corresponding Constrain functions are. In addition, the errorless setting allows PConstrain to be computed with good parallelism (i.e., in low depth) by an alternative algorithm that "short circuits" the computation via a base case that constrains over an *entire (sub)tree* in just one step. More specifically, we modify the base case (Lines 1 and 2) of Algorithm 1 as shown in Algorithm 2 below. The rest of the algorithm remains unchanged, apart from the fact that PConstrain does not take or output any error terms.

In Lemma 6.3.7 we prove that the alternative algorithm is correct. Then in Section 6.3.3.2 we describe how $\mathsf{PConstrain}$ can be evaluated in low depth.

Algorithm 2 $\mathsf{PConstrain}_{T,w,x}\colon \mathcal{V}_{T,w} \times \mathbb{Z}_q^n \to \mathcal{V}_{T,wx} \times \mathbb{Z}_q^n$ for $|wx| \leq |T|$, $x \neq \varepsilon$

Require: $(\mathbf{v}, \mathbf{s}) \in \mathcal{V}_{T,w} \times \mathbb{Z}_q^n$

1: **if** $w = \varepsilon$ and $|x| = |T|$ **then** ▷ base case
  return $\mathsf{Decode}(\mathbf{v} \cdot \mathbf{A}_T(x))$
2: **end if**
3: *The remaining code is the same as in Algorithm 1, but without any error terms* $(\mathbf{e}_y)$.

Lemma 6.3.7. For any fully binary tree T, any bit strings w, x with $|wx| \leq |T|$, and any $(\mathbf{v}, \mathbf{s}) \in \mathcal{V}_{T,w} \times \mathbb{Z}_q^n$,

$$\mathsf{PConstrain}_{T,w,x}(\mathbf{v}, \mathbf{s}) = \mathsf{Constrain}'_{T,w,x}(\mathbf{v}, \mathbf{s}, \mathbf{0}) \ . \tag{6.3.19}$$

Proof. Because the two algorithms differ only in the base case, it suffices to show that Equation (6.3.19) holds when $w = \epsilon$ and $|x| = |T|$. We prove this by induction on $|T|$.

If $|T| = 1$, then $\mathbf{A}_T(x) = \mathbf{A}_x$ and the claim holds by inspection. In the inductive case, we parse $x = x_l x_r$ with $|x_l| = |T.l|$, and adopt the notation from the computation of $\mathsf{Constrain}'_{T,\varepsilon,x}$. In this case, the induction hypothesis on $T.l$ implies that

$$(\mathbf{w}, \mathbf{t}) := \mathsf{Constrain}'_{T.l,\varepsilon,x_l}(\mathbf{s}, \mathbf{0}) = \mathsf{PConstrain}_{T.l,\epsilon,x_l}(\mathbf{s}) = \mathsf{Decode}(\mathbf{s} \cdot \mathbf{A}_{T.l}(x_l)) \ ,$$

where $(\mathbf{w}, \mathbf{t}) \in \mathcal{P} \times \mathbb{Z}_q^n$, as usual. By the induction hypothesis on $T.r$, we also have that

$$\mathsf{Constrain}'_{T.r,\epsilon,x_r}(\mathbf{t}, \mathbf{0}) = \mathsf{PConstrain}_{T.r,\epsilon,x_r}(\mathbf{t}) = \mathsf{Decode}(\mathbf{t} \cdot \mathbf{A}_{T.r}(x_r)) \ .$$

Combining the above, we obtain

$$\begin{aligned}
&\mathsf{Decode}^{-1}(\mathsf{Constrain}'_{T,\epsilon,x}(\mathbf{s},\mathbf{0})) \\
&\quad = \mathbf{w}\cdot\mathbf{G}^{-1}(\mathbf{A}_{T.r}(x_r)) + \mathsf{Decode}^{-1}(\mathsf{Constrain}'_{T.r,\epsilon,x_r}(\mathbf{t},\mathbf{0})) \\
&\quad = \mathbf{w}\cdot\mathbf{G}^{-1}(\mathbf{A}_{T.r}(x_r)) + \mathbf{t}\cdot\mathbf{A}_{T.r}(x_r) \\
&\quad = \mathsf{Decode}^{-1}(\mathbf{w},\mathbf{t})\cdot\mathbf{G}^{-1}(\mathbf{A}_{T.r}(x_r)) \\
&\quad = \mathbf{s}\cdot\mathbf{A}_{T.l}(x_l)\cdot\mathbf{G}^{-1}(\mathbf{A}_{T.r}(x_r)) = \mathbf{s}\cdot\mathbf{A}_T(x)\ ,
\end{aligned}$$

where the last line follows by the definition of $\mathbf{A}_T(\cdot)$. This completes the proof. □

6.3.3.2 Parallel Evaluation of PConstrain

We now analyze the parallel complexity of the PConstrain functions according to Algorithm 2 (and Algorithm 1) above. Our main goal is to bound what we call the "nonlinear depth" of $\mathsf{PConstrain}_{T,w,x}$ in terms of the topology of T and the strings w, x. Nonlinear depth only takes into account the nonlinear Decode and $\mathbf{G}^{-1}$ operations; the remaining operations are all linear over $\mathbb{Z}_q$. For an implementation of PConstrain by an arithmetic or boolean circuit, the depth will depend on the precise circuit model used and the implementation of the linear and nonlinear operations, but in any case the final depth will be proportional to the nonlinear depth.

To state our claim we recall from Chapter 5 the notions of "left depth" and "right depth" of the jth leaf in a binary tree T, and of T itself. The left depth $l_T(j)$ (respectively, right depth $r_T(j)$) of the jth leaf is the number of edges from a parent to its left (resp., right) child on the path from the root to that leaf. The left and right depths $l(T), r(T)$ are respectively the maximum left and right depths over all leaves in T.

Lemma 6.3.8. The function $\mathsf{PConstrain}_{T,w,x}(\mathbf{v},\mathbf{s})$ can be computed via (1) a preprocessing phase (independent of $(\mathbf{v},\mathbf{s})$) of nonlinear depth at most $r(T)$, and (2) an online phase (dependent on $(\mathbf{v},\mathbf{s})$) of nonlinear depth at most $l_T(|w|) + r_T(|x|) \leq$

$l(T) + r(T)$.

We remark that in Chapter 5, the nonlinear depth of computing the (non-constrained) PRF is just $r(T)$, so one can obtain an extremely parallel PRF using a "left spine" tree with $r(T) = 1$ and $l(T) = |T| - 1$ (this corresponds to the function from [18]). But here, evaluating the PRF from a constrained key can require nonlinear depth proportional to the sum of T's left and right depths. Therefore, to get good parallelism for all w, x we must use a shallow tree T, e.g., one with depth $O(\log|T|)$.

Proof. We first observe that in a call to $\mathsf{PConstrain}_{T,w,x}$ on a pair $(\mathbf{v}, \mathbf{s}) \in \mathcal{V}_{T,w} \times \mathbb{Z}_q^n$, all the $\mathbf{G}^{-1}(\mathbf{A}_{T'}(z))$ matrices that are required in the evaluation are independent of $(\mathbf{v}, \mathbf{s})$, and hence may be (pre)computed in parallel of everything else. As shown in Chapter 5, the nonlinear depth of computing $\mathbf{A}_{T'}(z)$ is the right depth of T', and in this case each T' is a subtree of T.

Now, assuming that the above $\mathbf{G}^{-1}(\cdot)$ matrices have been computed separately, we analyze the nonlinear depth of $\mathsf{PConstrain}_{T,w,x}$, which is due entirely to Decode operations. To perform this analysis we first introduce an optimization that eliminates some redundant nonlinear operations. Notice that in line 7, we apply $\mathsf{Decode}^{-1}(\mathbf{k})$ where $\mathbf{k}$ came from a recursive call that completed the right subtree, and hence is an output of Decode (returned from line 1 or 7). To eliminate this redundancy, in lines 1 and 7 we remove the applications of Decode and Decode^{-1}, and in line 11 we apply Decode to the result of the recursive call. (Also, we apply Decode to the ultimate output, if necessary.) With these changes, we have the following observations:

- In the base case $w = \varepsilon$ and $|wx| = |T|$, the nonlinear depth is zero.
- The case $|wx| < |T.l|$, where we just recurse into the left subtree, has the same nonlinear depth as its recursive call.
- The case $|T.l| \leq |w|$, where we just recurse into the right subtree and conditionally fold (using a linear operation), has the same nonlinear depth as its recursive

call.

- The case $|w| < |T.l| \leq |wx|$, where we recurse to complete the left subtree, then decode, then recurse into the right subtree, has nonlinear depth one more than the sum of the depths of the two recursive calls.

We therefore have only two interesting cases to analyze: (1) completing a tree from some arbitrary nonempty starting string, i.e., $0 < |w| < |wx| = |T|$, and (2) constraining from the empty string to some arbitrary incomplete string, i.e., $0 = |w| < |wx| < |T|$. In the case (1) we can see that the nonlinear depth is simply the "left depth" of the $|w|$th leaf of T, by the third and fourth observations above. Similarly, in case (2) we see that the nonlinear depth is the "right depth" of the $|x|$th leaf of T, by the second and third observations above. The lemma follows. □

6.3.4 "Rounded" Function Family $\overline{\mathcal{C}}$

We now define our final "rounded" family of constraining functions, denoted $\overline{\mathcal{C}}$, which we prove to be pseudorandom, as well as (computationally) key-homomorphic and consistent. In $\overline{\mathcal{C}}$ we use the same DAG on $\{0,1\}^{\leq |T|}$ as in the noisy function family, but we define somewhat different "rounded" (and errorless) key spaces, and thereby different constraining functions and group operations.

Rounding Factors and the Modulus. The family $\overline{\mathcal{C}}$ is parameterized by a "rounding factor" $e_{T'}$ for each subtree T' of T. For convenience of analysis, we choose these factors to all be powers of two. The factors are defined recursively to satisfy the inequalities

$$e_{T'} \geq \begin{cases} r \cdot \lambda^{\omega(1)} & \text{if } |T'| = 1, \\ (e_{T'.l} \cdot (n\ell) + e_{T'.r}) \cdot \lambda^{\omega(1)} & \text{otherwise.} \end{cases} \tag{6.3.23}$$

Using these rounding factors, we define $\lfloor \cdot \rfloor_T := \lfloor \cdot \rfloor_{e_T}$, for any tree T, to simply exposition further ahead.

Our proofs ahead require e_T for the underlying tree T of the function family to divide the modulus q, which is also chosen to be a power of 2. More concretely, we require $q \geq e_T \cdot \lambda^{\omega(1)}$. In particular, this implies that

$$q \geq r \cdot \lambda^{\omega(d(T))},$$

where $d(T)$ represents the maximum depth of T. The quantity r is a parameter of the function family. It corresponds to the parameter of the subgaussian distribution χ used in the function family $\mathcal{C}$ in Section 6.3.2 above.

Key Space and Distribution. The key space for $w \in \{0,1\}^{\leq |T|}$ and its associated distribution are simply defined to be

$$\overline{\mathcal{K}}_{T,w} := \mathbb{Z}_q^n,$$
$$\overline{\mathcal{D}}_{T,w} := U(\overline{\mathcal{K}}_{T,w}).$$

Auxiliary Space. Mirroring Equation (6.3.5), we define the "rounded" auxiliary domain $\overline{\mathcal{V}}_{T,w}$ for $w \in \{0,1\}^{\leq |T|}$ as follows:

$$\overline{\mathcal{V}}_{T,w} := \begin{cases} \{\varepsilon\} & \text{if } |w| = 0, \\ \overline{\mathcal{V}}_{T.l,w} & \text{if } 0 < |w| < |T.l|, \\ \lfloor \mathcal{P} \rfloor_{T.l} \times \overline{\mathcal{V}}_{T.r,w_r} & \text{if } |T.l| \leq |w| < |T|, \\ \lfloor \mathcal{P} \rfloor_T & \text{if } |w| = |T|, \end{cases} \tag{6.3.26}$$

where we parse $w = w_l w_r$ with $|w_l| = |T.l|$ in the third case above. Note that for every subtree T' of T, we have $\lfloor \mathcal{P} \rfloor_{T'} \subseteq \mathcal{P}$ (because every $e_{T'}$ divides q), and therefore $\overline{\mathcal{V}}_{T,w} \subseteq \mathcal{V}_{T,w}$.

Constraining functions. We first generalize the rounding operator $\lfloor \cdot \rfloor$ to work over inputs in $\mathcal{V}_{T,w}$, producing outputs in $\overline{\mathcal{V}}_{T,w}$ for $w \in \{0,1\}^{\leq |T|}$. As with the

addition operator $\oplus$, the parameters T and w will be apparent from the context. The definition mirrors Equation (6.3.26) above, as follows:

$$\lfloor \mathbf{v} \rfloor := \begin{cases} \epsilon & \text{if } j = 0, \\ \lfloor \mathbf{v} \rfloor & \text{if } 0 < |w| < |T.l|, \\ \left(\lfloor \mathbf{v}_l \rfloor_{T.l}, \lfloor \mathbf{v}_r \rfloor \right) & \text{if } |T.l| \leq |w| < |T|, \\ \lfloor \mathbf{v} \rfloor_T & \text{if } |w| = |T|. \end{cases} \tag{6.3.28}$$

where in the second case, we recurse over $T.l$, recasting the input in $\mathcal{V}_{T.l,w}$ and in the third case we parse $w = w_l w_r$ (with $|w_l| = |T.l|$), $\mathbf{v} = (\mathbf{v}_l, \mathbf{v}_r) \in \mathcal{P} \times \mathcal{V}_{T.r,w_r}$. For convenience, we also extend $\lfloor \cdot \rfloor$ to $\mathcal{V}_{T,w} \times \mathbb{Z}_q^n$, passing through the $\mathbb{Z}_q^n$ component to the output without modification.

The "rounded" constraining functions $\overline{\mathsf{PConstrain}}_{T,w,x} \colon \overline{\mathcal{V}}_{T,w} \times \overline{\mathcal{K}}_{T,w} \to \overline{\mathcal{V}}_{T,wx} \times \overline{\mathcal{K}}_{T,wx}$ are then defined simply as

$$\overline{\mathsf{PConstrain}}_{T,w,x}(\mathbf{k}) := \lfloor \mathsf{PConstrain}_{T,w,x}(\mathbf{k}) \rfloor \ . \tag{6.3.30}$$

6.3.4.1 Preliminaries

In this section, we give some more supporting definitions and claims that help us in proving the properties about $\overline{\mathcal{C}}$ that we want to, *viz.*, pseudorandomness, consistency and homomorphism.

The following fact follows easily from the definition of $\lfloor \cdot \rfloor$ in Equation (6.3.28).

Fact 6.3.9. For any $w \in \{0,1\}^{\leq |T|}$ and $\mathbf{v} \in \mathcal{V}_{T,w}$, let $\mathbf{d} = \lfloor \mathbf{v} \rfloor - \mathbf{v}$. Then the following are true about $\mathbf{d}$:

1. Every component of every $\mathcal{P}$ component (with associated tree T') $\mathbf{d}'$ of $\mathbf{d}$ lies in the interval $\{-e_{T'}, \ldots, 0\}$ and
2. For any $\mathbf{s}, \mathbf{t} \in \mathbb{Z}_q^n$, $(\mathbf{v}, \mathbf{s}) \oplus (\mathbf{d}, \mathbf{t}) = (\lfloor \mathbf{v} \rfloor, \mathbf{s} + \mathbf{t})$.

"Bad" predicates. To aid the proofs ahead, we define two predicates which help us in characterising certain "bad" elements of $\mathcal{V}_{T,w}$. For every $w \in \{0,1\}^{\leq |T|}$, $\mathsf{BAD}^1_{T,w,B}$ and $\mathsf{BAD}^2_{T,w,B}$, where $B = B_{T'}$ is a function from trees T' to positive integers$\mathbb{Z}^+$ (like $\Phi_{T'}$ or $e_{T'}$), are defined as follows.

1. For $\mathbf{v} \in \mathcal{V}_{T,w}$, $\mathsf{BAD}^1_{T,w,B}(\mathbf{v})$ is true if for some $\mathcal{P}$-component $\mathbf{v}'$ (of $\mathbf{v}$) associated with subtree T',

$$\left\lfloor \mathbf{v}' + \{-B_{T'}, \ldots, B_{T'}\}^{n\ell} \right\rfloor_{T'} \neq \{\lfloor \mathbf{v}' \rfloor_{T'}\}.$$

   In words, the predicate indicates that not all vectors in the neighborhood of $\mathbf{v}$ round to the same value. We note that given $\mathbf{v}$, it is efficient to test $\mathsf{BAD}^1_{T,w,B}(\mathbf{v})$ – the (complement of) test comprises simply checking if $\lfloor v'_i \pm B_{T'} \rfloor_{T'} = \lfloor v'_i \rfloor_{T'}$ on each co-ordinate v'_i of $\mathbf{v}'$, for $\mathcal{P}$ components $\mathbf{v}'$ with associated subtrees T'.

2. For $\mathbf{v} \in \mathcal{V}_{T,w}$, $\mathsf{BAD}^2_{T,w,B}(\mathbf{v})$ is true if for some $\mathcal{P}$-component $\mathbf{v}'$ (of $\mathbf{v}$) associated with subtree T',

$$\mathbf{v}' + \{-B_{T'}, \ldots, B_{T'}\}^{n\ell} \notin \mathcal{P}.$$

   In words, the predicate indicates that $\mathbf{v}$ is "too close" to the border of $\mathcal{P}$. The efficient test for (the complement of) $\mathsf{BAD}^2_{T,w,B}(\mathbf{v})$ comprises checking if $-q/4 + B_{T'} \leq v'_i < q/4 - B_{T'}$ on each co-ordinate v'_i of $\mathbf{v}'$, for $\mathcal{P}$ components $\mathbf{v}'$ with associated subtrees T'.

As with $\lfloor \cdot \rfloor$, we extend the BAD predicates to accept inputs in $\mathcal{V}_{T,w} \times \mathbb{Z}_q^n$, ignoring the $\mathbb{Z}_q^n$ component of the input. Note that $\mathsf{BAD}^1_{T,w,B}(\mathbf{v})$ being false immediately implies that $\mathsf{BAD}^2_{T,w,B}(\mathbf{v})$ is false as well.

Bounding Errors and Characterizing Bad Predicates. We need to analyze the accumulated error in fully constrained keys over arbitrary trees. For this purpose we define a "growth factor" Φ_T associated with an arbitrary full binary tree, defined

recursively as follows:

$$\Phi_T := \begin{cases} 1 & \text{if } |T| = 1, \\ \sqrt{(\Phi_{T.l} \cdot n\ell)^2 + (\Phi_{T.r})^2} & \text{otherwise.} \end{cases} \tag{6.3.32}$$

Note that by inspection of Equations (6.3.32) and (6.3.23), for all subtrees T' we have

$$e_{T'} \geq r \cdot \Phi_{T'} \cdot \lambda^{\omega(1)} \ . \tag{6.3.34}$$

We next prove a lemma that is essentially a restatement of Lemma 5.3.6.

Lemma 6.3.10 (Error Bound). Let T be a full binary tree and let $w \in \{0,1\}^{\leq |T|}$. Then if $q \geq 4r \cdot \Phi_T \cdot \omega(\sqrt{\log \lambda})$, the following are true about $\mathsf{Constrain}'_{T,\epsilon,w}(\mathbf{0}, (\mathbf{e}_y))$ with $1 - \mathrm{negl}(\lambda)$ probability over the choice of $(\mathbf{e}_y) \leftarrow \mathcal{W}_{T,0}$: (1) its $\mathbb{Z}_q^n$ component is zero, and (2) each $\mathcal{P}$-component, with associated subtree T', is subgaussian with parameter $r \cdot \Phi_{T'}$. The same conclusions hold with overwhelming probability for the output of $\mathsf{Constrain}'_{T,w,x}(\mathbf{0}, \mathbf{0}, (\mathbf{e}_y))$ for nonempty $w, x \in \{0,1\}^*$ such that $|wx| \leq |T|$.

Proof. To prove the first claim, we proceed by induction on $|T|$. Note that the claim is trivially true in the case $w = \epsilon$. In the base case, the output is $\mathsf{Decode}(\mathbf{e}_w)$. By hypothesis on q, we have that $\|\mathbf{e}_w\|_\infty \leq q/4$ and therefore $\mathbf{e}_w \in \mathcal{P}$ with overwhelming probability. Note that $\Phi_T > \Phi_{T'}$ for all subtrees T' of T and therefore the hypothesis on q holds for all inductive cases. The case that $|w| < |T.l|$ follows immediately by the inductive hypothesis and definition of $\mathsf{Constrain}'_{T,\varepsilon,w}$.

If $|T.l| \leq |w|$, then parse $w = w_l w_r$ for $|w_l| = |T.l|$. In this case we know by hypothesis that $\mathsf{Constrain}'_{T.l,\epsilon,w_l}(\mathbf{0}, (\mathbf{e}_y)_{|y| \leq |T.l|})$ has a $\mathcal{P}$-component that is subgaussian with parameter $r \cdot \Phi_{T.l}$ and a zero $\mathbb{Z}_q^n$ component with overwhelming probability. Conditioned on this happening, we recurse into the right subtree, where we know that the output of $\mathsf{Constrain}'_{T.r,\epsilon,w_r}(\mathbf{0}, (\mathbf{e}_{w_l y})_{|y| \leq |T.r|})$ is of the correct form with overwhelming probability by hypothesis. Now, if $|w| < |T|$, then we prepend the $\mathcal{P}$-component

output of the computation over $T.l$ to this output, obtaining the complete output in the desired form.

On the other hand, if $|w| = |T|$, then $\mathsf{Decode}^{-1}(\mathsf{Constrain}'_{T.r,\epsilon,w_r}(\mathbf{0}, (\mathbf{e}_{w_l y})_{|y|\leq|T.r|}))$ is subgaussian with parameter $r \cdot \Phi_{T.r}$. By Lemma 2.1.1 on the spectral norm of binary matrices, the definition of Φ_T in terms of $\Phi_{T.l}$ and $\Phi_{T.r}$ and the Euclidean additivity of subgaussians, we have that the $\mathsf{Decode}^{-1}(\cdot)$ of the final $\mathsf{Constrain}'_{T,\epsilon,w}$ output is subgaussian with parameter $r \cdot \Phi_T$. By assumption on q, this lies completely in $\mathcal{P}$ with high probability, and therefore the decoded output is as desired. This completes the proof of the first claim.

To prove the second claim, we firstly note by the first claim that with overwhelming probability, $\mathsf{Constrain}'_{T,\epsilon,w}(\mathbf{0}, (\mathbf{e}_y)) = (\mathbf{d}, \mathbf{0})$ for some $\mathbf{d} \in \mathcal{V}_{T,w}$ with subgaussian components. We also know that the output of $\mathsf{Constrain}'_{T,\epsilon,wx}(\mathbf{0}, (\mathbf{e}_y))$ is also of similar form. However, by consistency, we have that

$$\begin{aligned}\mathsf{Constrain}'_{T,\epsilon,wx}(\mathbf{0}, (\mathbf{e}_y)) &= \mathsf{Constrain}'_{T,w,x}(\mathsf{Constrain}'_{T,\epsilon,w}(\mathbf{0}, (\mathbf{e}_y)_{|y|\leq|w|}), (\mathbf{e}_{wy})_{|y|\leq|T|-|w|}) \\ &= \mathsf{Constrain}'_{T,w,x}(\mathbf{d}, \mathbf{0}, (\mathbf{e}_{wy})_{|y|\leq|T|-|w|}) \ .\end{aligned}$$

Now, we know that with overwhelming probability, $\mathsf{Constrain}'_{T,w,x}(\mathbf{d}, \mathbf{0}, (\mathbf{e}_y))$ is subgaussian in all $\mathcal{P}$-components, and has zero $\mathbb{Z}_q^n$ component, with a subgaussian input $\mathbf{d} \in \mathcal{V}_{T,w}$. It follows that if $\mathbf{d}$ were instead zero, the output would only fall in magnitude, remaining subgaussian. □

We use the fact above to claim that the rounded versions of $\mathsf{Constrain}'$ with errors and $\mathsf{PConstrain}$ are the same, with a few caveats. This is used in the proof of pseudorandomness.

Claim 6.3.11. Let $w \in \{0,1\}^{\leq|T|}$ be arbitrary, let $R_{T'} = r \cdot \Phi_{T'} \cdot \omega(\sqrt{\log \lambda})$ and let $(\mathbf{s}, (\mathbf{e}_y)) \leftarrow \mathcal{D}_{T,\epsilon}$, where r is the parameter of the discrete gaussian χ used to define the distribution $\mathcal{W}_{T,0}$. If $\mathsf{BAD}^1_{T,w,R_{T'}}(\mathsf{Constrain}'_{T,\epsilon,w}(\mathbf{s}, (\mathbf{e}_y)))$ is false, then with

overwhelming $(1 - \mathrm{negl}(\lambda))$ probability, it is the case that

$$\lfloor \mathsf{Constrain}'_{T,\epsilon,w}(\mathbf{s}, (\mathbf{e}_y)) \rfloor = \overline{\mathsf{PConstrain}}_{T,\epsilon,w}(\mathbf{s}) \ .$$

Proof. Recall that by definition, $\overline{\mathsf{PConstrain}}_{T,\varepsilon,w}(\mathbf{s}) = \lfloor \mathsf{Constrain}'_{T,\epsilon,w}(\mathbf{s}, \mathbf{0}) \rfloor$. We let $\mathbf{k} = \mathsf{Constrain}'_{T,\epsilon,w}(\mathbf{s}, (\mathbf{e}_y))$, $\mathbf{f} = \mathsf{Constrain}'_{T,\epsilon,w}(\mathbf{s}, \mathbf{0})$ and $\mathbf{d} = \mathsf{Constrain}'_{T,\epsilon,w}(\mathbf{0}, (\mathbf{e}_y))$. It follows by homomorphism of $\mathsf{Constrain}'$ (Lemma 6.3.3) that $\mathbf{k} = \mathbf{f} \oplus \mathbf{d}$. By Lemma 6.3.10, we have that the $\mathbb{Z}_q^n$ component of $\mathbf{d}$ is zero, and every $\mathcal{P}$-component $\mathbf{d}'$ (with associated subtree T') is subgaussian with parameter $r \cdot \Phi_{T'}$, and thus $\|\mathbf{d}'\|_\infty \leq R_{T'}$ with an overwhelming probability. Since $\mathsf{BAD}^1_{T,w,R_{T'}}(\mathbf{k})$ is false, we have that $\lfloor \mathbf{k}' + \{-R_{T'}, \ldots, R_{T'}\}^{n\ell} \rfloor_{T'} = \{\lfloor \mathbf{k} \rfloor\}_{T'}$ for each $\mathcal{P}$-component $\mathbf{k}'$ (with associated tree T'), and it follows that $\mathbf{k}' + \{-R_{T'}, \ldots, R_{T'}\}^{n\ell} \in \mathcal{P}$. The previous two sentences imply that $\mathbf{k} = \mathbf{f} + \mathbf{d}$, where the addition here happens over $\mathbb{Z}$. Moreover, since the $\mathbb{Z}_q^n$ component of $\mathbf{d}$ is zero, we also have that the $\mathbb{Z}_q^n$ components of $\mathbf{k}$ and $\mathbf{f}$ are equal. It follows that $\lfloor \mathbf{f} \rfloor = \lfloor \mathbf{k} - \mathbf{d} \rfloor = \lfloor \mathbf{k} \rfloor$, and this proves the claim. □

Claim 6.3.12. Let $w \in \{0,1\}^{\leq |T|}$ and let $\mathbf{v} \leftarrow U(\mathcal{V}_{T,w})$.

1. Let $B_{T'}$ be such that $e_{T'} \geq B_{T'} \cdot \lambda^{\omega(1)}$ for every subtree T' of T. Then the following holds.
$$\Pr\left[\mathsf{BAD}^1_{T,w,B_{T'}}(\mathbf{v})\right] = \mathrm{negl}(\lambda) \ . \tag{6.3.37}$$
2. Let $B_{T'}$ be such that $q/4 \geq B_{T'} \cdot \lambda^{\omega(1)}$ for every subtree T' of T. Then the following holds.
$$\Pr\left[\mathsf{BAD}^2_{T,w,B_{T'}}(\mathbf{v})\right] = \mathrm{negl}(\lambda) \ . \tag{6.3.39}$$

Proof. We first prove the validity of Equation (6.3.37). By a simple counting argument, it is easy to see that if $v \leftarrow U(\{-q/4, \ldots, q/4 - 1\})$, then the following holds for any T' that is a subtree of T.

$$\Pr[\lfloor v \pm B_{T'} \rfloor_{T'} \neq \lfloor v \rfloor_{T'}] = \frac{B_{T'}}{e_{T'}} = \mathrm{negl}(\lambda) \ , \tag{6.3.41}$$

where the last equality comes from the fact that $e_{T'} \geq B_{T'} \cdot \lambda^{\omega(1)}$, by hypothesis. The first result follows by simple union bounds over the individual $\mathcal{P}$ components of $\mathbf{v}$ and the co-ordinates of each component, allied with Equation (6.3.41) above.

The validity of Equation (6.3.39) is proven in exactly the same way, except that instead of Equation (6.3.41), we have the following equation.

$$\Pr[v \pm B_{T'} \notin \{-q/4, \ldots, q/4 - 1\}] = \frac{B_{T'}}{q/4} = \mathrm{negl}(\lambda) \ ,$$

where the last equality comes from the fact that $q/4 \geq B_{T'} \cdot \lambda^{\omega(1)}$, by hypothesis. $\square$

6.3.4.2 Pseudorandomness

We now show that the construction of the family $\overline{\mathcal{C}}$ from Section 6.3.4 is a constrained PRF, according to Definition 6.2.2. Here, we prove pseudorandomness of the function as defined in Definition 6.2.2, and use the Security of the $\mathsf{Constrain}$ family of functions, as defined in Section 6.3.2 above.

Theorem 6.3.13 (Pseudorandomness). *The construction described in Section 6.3.4 is pseudorandom according to Definition 6.2.2 for the set of challenge nodes* $R = \{0,1\}^{\leq |T|}$ *under the* $\mathsf{LWE}_{n,q,\chi}$ *assumption, where* $\chi = D_{\mathbb{Z},r}$ *and* $q \geq e_T$.

Proof. We proceed though a series of games, as detailed below.

Game H_0. This is the "real" world game, as described in Definition 6.2.2. To recall, the root of the underlying DAG is initialized with a key $\mathbf{s} \leftarrow \overline{\mathcal{D}}_{T,\epsilon}$. The adversary is allowed to (adaptively) query $\overline{\mathsf{PConstrain}}_{T,\epsilon,w}$ such that no query is a prefix of any other (this is equivalent to the corresponding nodes in the underlying DAG structure having no common descendant).

Game H_1. Here we (lazily) sample the key $(\mathbf{s}, (\mathbf{e}_y)) \leftarrow \mathcal{D}_{T,\epsilon}$ to initialize the root of the DAG. For any legal query node w, we compute $(\mathbf{w}, \mathbf{t}) = \mathsf{Constrain}'_{T,\epsilon,w}(\mathbf{s}, (\mathbf{e}_y))$,

and return $\lfloor (\mathbf{w}, \mathbf{t}) \rfloor$, if $\mathsf{BAD}^1_{T,w,R_{T'}}(\mathbf{w})$ is false, where $R_{T'} = r \cdot \Phi_{T'} \cdot \omega(\sqrt{\log \lambda})$. We abort and abandon the experiment if the predicate evaluates to true. In Claim 6.3.11, we show that with $1 - \mathrm{negl}(\lambda)$ probability over the choice of $(\mathbf{e}_y) \leftarrow \mathcal{W}_{T,0}$, as long as $\mathsf{BAD}^1_{T,w,R_{T'}}$ is not satisfied, it is true that $\lfloor (\mathbf{w}, \mathbf{s}) \rfloor = \overline{\mathsf{PConstrain}}_{T,\epsilon,w}(\mathbf{s})$. It follows that for any (potentially unbounded) attacker $\mathcal{A}$,

$$\begin{aligned}\mathbf{Adv}_{H_0,H_1}(\mathcal{A}) \leq & \Pr[\mathsf{BAD}^1_{T,w,R_{T'}}(\mathbf{w}) \text{ is true in } H_1 \text{ with attacker } \mathcal{A} \text{ for some } w] \\ & + \mathrm{negl}(\lambda) \ . \end{aligned} \tag{6.3.43}$$

We do not directly bound the probability of $\mathsf{BAD}^1_{T,w,R_{T'}}$ happening in H_1, instead deferring analysis to a future game, where we prove that this is indeed negligible.

Game H_2. This game works similarly to the previous one, except that we sample $\mathbf{t} \leftarrow U(\mathbb{Z}_q^n)$ and return $\lfloor \mathbf{w}, \mathbf{t} \rfloor$ on all legal queries w, aborting instead if $\mathsf{BAD}^1_{T,w,R_{T'}}(\mathbf{w})$ is true, where $\mathbf{w}$ is computed as before.

We notice that Games H_1 and H_2 are exactly the real and ideal games respectively for the family $\mathcal{C}$, except that the output corresponding to each query w is projected on to the relevant $\mathcal{V}_{T,w} \times \mathbb{Z}_q^n$ domain, the bad event (and subsequent abort) is tested, and then the appropriate rounding operation $\lfloor \cdot \rfloor$ is applied. Since $\mathcal{C}$ is pseudorandom under the LWE assumption from the hypothesis by Theorem 6.3.6; and since the operation $\lfloor \cdot \rfloor$ is efficient to apply, and the $\mathsf{BAD}^1_{T,w,R_{T'}}$ predicate is also efficient to test, a trivial simulation shows us that for any computationally efficient attacker $\mathcal{A}$, $\mathbf{Adv}_{H_1,H_2}(\mathcal{A}) = \mathrm{negl}(\lambda)$. For the same reasons, it also follows that for any efficient attacker $\mathcal{A}$

$$\begin{aligned}&|\Pr[\text{Some } \mathsf{BAD}^1_{T,w,R_{T'}}(\mathbf{w}) \text{ is true in } H_1] - \Pr[\text{Some } \mathsf{BAD}^1_{T,w,R_{T'}}(\mathbf{w}) \text{ is true in } H_2]| \\ &\quad = \mathrm{negl}(\lambda) \ . \end{aligned} \tag{6.3.44}$$

We now try to bound the quantity $\Pr[\mathsf{BAD}^1_{T,w,R_{T'}}(\mathbf{w})$ is true in H_2 with attacker $\mathcal{A}$ for some query $w]$. By Equation (6.3.34), we can infer that $e_{T'} \geq R_{T'} \cdot \lambda^{\omega(1)}$. In

H_2, if $\mathbf{w} \leftarrow U(\mathcal{V}_{T,w})$ is uniformly random, for any particular query w, we can use Claim 6.3.12 to establish that $\mathsf{BAD}^1_{T,w,R_{T'}}(\mathbf{w})$ is true only with negligible probability. We also know that if we fix the query w, then by Corollary 6.3.5 (which holds true under the LWE assumption from the hypothesis), $\mathbf{w}$ is *pseudorandom*. Since $\mathcal{A}$ is efficient, a straightforward hybrid simulation proves that $\Pr[\mathsf{BAD}^1_{T,w,R_{T'}}(\mathbf{w})$ is true in H_2 with attacker $\mathcal{A}$ for the fixed query $w] = \mathrm{negl}(\lambda)$. By a union bound over all the (polynomial) queries of $\mathcal{A}$, we know that $\Pr[\mathsf{BAD}^1_{T,w,R_{T'}}(\mathbf{w})$ is true in H_2 with attacker $\mathcal{A}$ for some query $w] = \mathrm{negl}(\lambda)$ and it follows by Equation (6.3.44) that $\Pr[\mathsf{BAD}^1_{T,w,R_{T'}}(\mathbf{w})$ is true in H_1 with attacker $\mathcal{A}$ for some query $w] = \mathrm{negl}(\lambda)$. This, allied with Equation (6.3.43), establishes that for any efficient attacker $\mathcal{A}$, $\mathbf{Adv}_{H_0,H_1}(\mathcal{A}) = \mathrm{negl}(\lambda)$.

Game H_3. In this game, for query w, we return $(\overline{\mathbf{w}}, \mathbf{t})$, where $\overline{\mathbf{w}}$ is computed as usual (that is, according to $\overline{\mathsf{PConstrain}}_{T,\epsilon,w}(\mathbf{s})$), and $\mathbf{t} \leftarrow \overline{\mathcal{D}}_{T,w}$. This is the "ideal" world game, as described in Definition 6.2.2. By Claim 6.3.11, we know that if $\mathsf{BAD}^1_{T,w,R_{T'}}$ is not satisfied in H_2, it is true that $\lfloor \mathsf{Constrain}'_{T,\epsilon,w}(\mathbf{s}, (\mathbf{e}_y)) \rfloor = \overline{\mathsf{PConstrain}}_{T,\epsilon,w}(\mathbf{s})$. Thus, it follows that for any efficient attacker $\mathcal{A}$,

$$\mathbf{Adv}_{H_2,H_3}(\mathcal{A}) \leq \Pr[\text{Some } \mathsf{BAD}^1_{T,w,R_{T'}}(\mathbf{w}) \text{ is true in } H_2 \text{ with attacker } \mathcal{A}] = \mathrm{negl}(\lambda) \ .$$

By the triangle inequality, we have for any efficient $\mathcal{A}$, $\mathbf{Adv}_{H_0,H_3}(\mathcal{A}) = \mathrm{negl}(\lambda)$. This completes the proof. □

6.3.4.3 Consistency and Homomorphism

We next show that the family $\overline{\mathcal{C}}$ from Section 6.3.4 is consistent and homomorphic, using the respective properties, along with the *security*, of the underlying $\mathcal{C}$ family of functions.

Lemma 6.3.14 (Consistency). For any full binary tree T, public parameters $\mathbf{A}_0, \mathbf{A}_1$,

bit-strings w, x, z, where $|wxz| \leq |T|$, and $\mathbf{v} \in \overline{\mathcal{V}}_{T,w}$, we have that with $1-\mathsf{negl}(\lambda)$ probability over the choice of the keys $\mathbf{s} \leftarrow \overline{\mathcal{D}}_{T,w}$,

$$\overline{\mathsf{PConstrain}}_{T,wx,z}(\overline{\mathsf{PConstrain}}_{T,w,x}(\mathbf{v},\mathbf{s})) = \overline{\mathsf{PConstrain}}_{T,w,xz}(\mathbf{v},\mathbf{s}) \ , \tag{6.3.46}$$

under the $\mathsf{LWE}_{n,q,\chi}$ assumption, where $\chi = D_{\mathbb{Z},r}$ and $q \geq e_T$.

Proof. Firstly, by Lemma 6.3.3 and the definition of $\mathsf{PConstrain}$ as $\mathsf{Constrain}'$ with zero errors, we can claim the following for *any* $w, x \in \{0,1\}^*$ such that $|wx| \leq |T|$, $\mathbf{v} \in \overline{\mathcal{V}}_{T,w}$, $\mathbf{s} \in \mathbb{Z}_q^n$ and errors $(\mathbf{e}_y) \in \mathcal{E}_{T,|w|}$:

$$\mathsf{PConstrain}_{T,w,x}(\mathbf{v},\mathbf{s}) = \mathsf{Constrain}'_{T,w,x}(\mathbf{v},\mathbf{s},(\mathbf{e}_y)) \oplus \mathsf{Constrain}'_{T,w,x}(\mathbf{0},\mathbf{0},(-\mathbf{e}_y)) \ . \tag{6.3.48}$$

By expanding $\overline{\mathsf{PConstrain}}_{T,w,x}(\cdot) = \lfloor \mathsf{PConstrain}_{T,w,x}(\cdot) \rceil$ (and similarly for other bit-strings) by Equation (6.3.30), and using the consistency property of $\mathsf{PConstrain}$ (see the discussion in Section 6.3.3.1), we can rewrite Equation (6.3.46) as follows

$$\begin{aligned} &\Big\lfloor \mathsf{PConstrain}_{T,wx,z}\big(\lfloor \mathsf{PConstrain}_{T,w,x}(\mathbf{v},\mathbf{s}) \rceil\big) \Big\rceil \\ &\qquad \stackrel{?}{=} \ \Big\lfloor \mathsf{PConstrain}_{T,wx,z}\big(\mathsf{PConstrain}_{T,w,x}(\mathbf{v},\mathbf{s})\big) \Big\rceil \ . \end{aligned} \tag{6.3.49}$$

Without loss of generality, we can assume that there is no "pass-through" in the execution of $\mathsf{PConstrain}_{T,w,xz}$ (and thus, also in $\mathsf{Constrain}'_{T,w,xz}$). This is because any $\mathcal{P}$ components passed through will be untouched by the computations happening on both the sides of Equation (6.3.46). By a similar inspection of Equation (6.3.49) and the idempotence of the rounding operation $\lfloor \cdot \rceil_{T'}$ over $\mathcal{P}$, we can also assume without loss of generality that there is no pass-through in the execution of $\mathsf{PConstrain}_{T,w,x}$ and $\mathsf{PConstrain}_{T,wx,z}$ (and their respective $\mathsf{Constrain}'$ counterparts).

By applying the observation in Equation (6.3.48) to the left hand side of Equation (6.3.49), we get the following, for any $(\mathbf{e}_y) \in \mathcal{E}_{T,|w|}$:

$$\begin{aligned}
&\Big\lfloor \mathsf{PConstrain}_{T,wx,z}\big(\lfloor \mathsf{PConstrain}_{T,w,x}(\mathbf{v},\mathbf{s})\rfloor\big)\Big\rfloor \\
&= \Big\lfloor \mathsf{Constrain}'_{T,wx,z}\big(\lfloor \mathsf{Constrain}_{T,w,x}(\mathbf{v},\mathbf{s},(\mathbf{e}_y)\oplus \mathsf{Constrain}_{T,w,x}(\mathbf{0},\mathbf{0},(-\mathbf{e}_y)))\rfloor\big)\Big\rfloor \\
&= \Big\lfloor \mathsf{Constrain}'_{T,wx,z}\big(\mathsf{Constrain}_{T,w,x}(\mathbf{v},\mathbf{s},(\mathbf{e}_y))\oplus \mathsf{Constrain}_{T,w,x}(\mathbf{0},\mathbf{0},(-\mathbf{e}_y))\oplus \mathbf{d}_{wx}\big)\Big\rfloor \\
&= \Big\lfloor \mathsf{Constrain}'_{T,w,xz}(\mathbf{v},\mathbf{s},(\mathbf{e}_y))\oplus \mathsf{Constrain}'_{T,wx,z}\big(\mathsf{Constrain}_{T,w,x}(\mathbf{0},\mathbf{0},(-\mathbf{e}_y))\oplus \mathbf{d}_{wx}\big)\Big\rfloor,
\end{aligned}$$

for some rounding offset $\mathbf{d}_{wx}$, where the second equation follows from Fact 6.3.9 and the last follows by the consistency (Lemma 6.3.1) and homomorphism (Lemma 6.3.3) of $\mathsf{Constrain}'$. If $(\mathbf{e}_y) \leftarrow \mathcal{W}_{T,|w|}$, by an application of Lemma 6.3.10, we know that every $\mathcal{P}$ component (with associated subtree T') of $\mathsf{Constrain}_{T,w,x}(\mathbf{0},\mathbf{0},(-\mathbf{e}_y))$ is subgaussian with parameter $r\cdot\Phi_{T'}$ and its $\mathbb{Z}_q^n$ component is zero with all but $\mathrm{negl}(\lambda)$ probability. Thus, we have that every $\mathcal{P}$ component of $\mathsf{Constrain}_{T,w,x}(\mathbf{0},\mathbf{0},(-\mathbf{e}_y))\oplus\mathbf{d}_{wx}$ is at most $e_{T'} + r\cdot\Phi_{T'}\cdot\omega(\sqrt{\log\lambda})$ in magnitude in every co-ordinate, and its $\mathbb{Z}_q^n$ component is zero, with overwhelming probability. Since there are no pass-throughs, we have by the definition of $e_{T'}$ in Equation (6.3.23) (in particular, the fact that $e_{T'} \geq (e_{T'.l}\cdot(n\ell) + e_{T'.r})\cdot\lambda^{\omega(1)}$) that every $\mathcal{P}$ component of $\mathsf{Constrain}'_{T,wx,z}\big(\mathsf{Constrain}_{T,w,x}(\mathbf{0},\mathbf{0},(-\mathbf{e}_y))\oplus \mathbf{d}_{wx}\big)$ is bounded in magnitude in every co-ordinate by some $B_{T'}$ such that $e_{T'} \geq B_{T'}\cdot\lambda^{\omega(1)}$, and its $\mathbb{Z}_q^n$ component is zero with overwhelming probability.

We can similarly apply the observation in Equation (6.3.48) to the right hand side of Equation (6.3.49), and apply homomorphism of $\mathsf{Constrain}'$ to obtain the following, for any $(\mathbf{e}_y) \in \mathcal{E}_{T,|w|}$:

$$\begin{aligned}
&\Big\lfloor \mathsf{PConstrain}_{T,wx,z}\big(\mathsf{PConstrain}_{T,w,x}(\mathbf{v},\mathbf{s})\big)\Big\rfloor \\
&\qquad = \Big\lfloor \mathsf{Constrain}'_{T,w,xz}(\mathbf{v},\mathbf{s},(\mathbf{e}_y))\oplus \mathsf{Constrain}'_{T,w,xz}(\mathbf{0},\mathbf{0},(-\mathbf{e}_y))\Big\rfloor\ .
\end{aligned}$$

Again, if $(\mathbf{e}_y) \leftarrow \mathcal{W}_{T,|w|}$, by an application of Lemma 6.3.10, we have that every $\mathcal{P}$ component (with associated subtree T') of $\mathsf{Constrain}_{T,w,xz}(\mathbf{0},\mathbf{0},-(\mathbf{e}_y))$ is bounded

in magnitude by $r \cdot \Phi_{T'} \cdot \omega(\sqrt{\log \lambda})$ in every co-ordinate and its $\mathbb{Z}_q^n$ component is zero with overwhelming probability.

Based on what we have above, we get the following system of equations,

$$\begin{aligned}
&\Pr_{\mathbf{s} \leftarrow \overline{\mathcal{D}}_{T,w}} [\overline{\mathsf{PConstrain}}_{T,wx,z}(\overline{\mathsf{PConstrain}}_{T,w,x}(\mathbf{v}, \mathbf{s})) \neq \overline{\mathsf{PConstrain}}_{T,w,xz}(\mathbf{v}, \mathbf{s})] \\
&= \Pr_{(\mathbf{s},(\mathbf{e}_y))} \Big[\Big\lfloor \mathsf{Constrain}'_{T,w,xz}(\mathbf{v}, \mathbf{s}, (\mathbf{e}_y)) \oplus \mathsf{Constrain}'_{T,wx,z}\big(\mathsf{Constrain}_{T,w,x}(\mathbf{0}, \mathbf{0}, (-\mathbf{e}_y)) \oplus \mathbf{d}_{wx}\big) \Big\rfloor \\
&\qquad \neq \Big\lfloor \mathsf{Constrain}'_{T,w,xz}(\mathbf{v}, \mathbf{s}, (\mathbf{e}_y)) \oplus \mathsf{Constrain}'_{T,w,xz}(\mathbf{0}, \mathbf{0}, (-\mathbf{e}_y)) \Big\rfloor \Big] \\
&\leq \Pr_{(\mathbf{s},(\mathbf{e}_y))} \Big[\mathsf{BAD}^1_{T,wxz,B_{T'}}\big(\mathsf{Constrain}'_{T,w,xz}(\mathbf{v}, \mathbf{s}, (\mathbf{e}_y))\big) \Big] + \mathrm{negl}(\lambda),
\end{aligned}$$

where the inequality follows from the fact that $\mathsf{BAD}^1_{T,wxz,B_{T'}}(\mathsf{Constrain}'_{T,w,xz}(\mathbf{v}, \mathbf{s}, (\mathbf{e}_y))$ is false, then both the left and the right hand sides evaluate to $\lfloor \mathsf{Constrain}'_{T,w,xz}(\mathbf{v}, \mathbf{s}, (\mathbf{e}_y)) \rfloor$ with overwhelming probability, since the $\mathbb{Z}_q^n$ components of the additive terms are zero and the $\mathcal{P}$ components are bounded in magnitude by $B_{T'} \geq r \cdot \Phi_{T'} \cdot \omega(\sqrt{\log \lambda})$ in every co-ordinate with overwhelming probability.

Now, if we consider efficient adversaries, we can replace $\mathsf{Constrain}'_{T,w,xz}(\mathbf{v}, \mathbf{s}, (\mathbf{e}_y))$ with a uniformly chosen $\mathbf{u} \leftarrow U(\mathcal{V}_{T,wxz} \times \mathbb{Z}_q^n)$, whilst incurring only a negligible penalty, by Corollary 6.3.5 (which is true under the LWE assumption from the hypothesis), remembering the fact that each of the $\mathcal{P}$ components in the output are freshly generated without loss of generality and the fact that BAD^1 is efficient to test. Therefore, for all efficient adversaries, the above probability expression evaluates as follows

$$\Pr_{(\mathbf{s},(\mathbf{e}_y))} \Big[\mathsf{BAD}^1_{T,wxz,B_{T'}}\big(\mathsf{Constrain}'_{T,w,xz}(\mathbf{v}, \mathbf{s}, (\mathbf{e}_y))\big) \Big] \leq \Pr_{\mathbf{u}} \Big[\mathsf{BAD}^1_{T,wxz,B_{T'}}(\mathbf{u}) \Big] \leq \mathrm{negl}(\lambda) \ ,$$

where the final inequality comes from the fact that $e_{T'} \geq B_{T'} \cdot \lambda^{\omega(1)}$ and Claim 6.3.12.$\square$

Before we prove the corresponding claim for homomorphism, we have to define an addition operation in the domain $\overline{\mathcal{V}}_{T,w} \times \overline{\mathcal{K}}_{T,w}$. Note that this cannot simply be $\oplus$,

because $\mathsf{Decode}(\mathbf{v}_1 + \mathbf{v}_2)$ for $\mathbf{v}_1, \mathbf{v}_2 \in \lfloor \mathcal{P} \rfloor_{T'}$ (for some left subtree T') may return a $\mathcal{P}$ element that is not in $\lfloor \mathcal{P} \rfloor_{T'}$ anymore. To deal with this, we define $\overline{\oplus}_{\mathbf{t}}$, analogous to $\oplus_{\mathbf{t}}$, which also takes in an auxiliary input $\mathbf{t} \in \mathbb{Z}_q^n$, in addition to two $\overline{\mathcal{V}}_{T,w} \times \overline{\mathcal{K}}_{T,w}$ elements. As with $\oplus$, we suppress the $\mathbf{t}$ term if it is zero. $\overline{\oplus}_{\mathbf{t}}$ is defined simply as

$$\mathbf{k}_1 \; \overline{\oplus}_{\mathbf{t}} \mathbf{k}_2 = \lfloor \mathbf{k}_1 \oplus_{\mathbf{t}} \mathbf{k}_2 \rfloor \ .$$

Lemma 6.3.15 (Homomorphism). For any full binary tree T, parameters $\mathbf{A}_0, \mathbf{A}_1$, strings w, x where $|wx| \leq |T|$ and auxiliaries $\mathbf{v}_1, \mathbf{v}_2 \in \overline{\mathcal{V}}_{T,w}$, we have that with $1 - \mathrm{negl}(\lambda)$ probability over the choice of the (constrained) keys $\mathbf{s}_1, \mathbf{s}_2 \leftarrow \overline{\mathcal{D}}_{T,w}$ that

$$\begin{aligned} &\overline{\mathsf{PConstrain}}_{T,w,x}(\mathbf{v}_1, \mathbf{s}_1) \; \overline{\oplus} \; \overline{\mathsf{PConstrain}}_{T,w,x}(\mathbf{v}_2, \mathbf{s}_2) \\ &\qquad = \overline{\mathsf{PConstrain}}_{T,w,x}((\mathbf{v}_1, \mathbf{s}_1) \; \overline{\oplus} \; (\mathbf{v}_2, \mathbf{s}_2)) + \mathbf{d} \ , \end{aligned} \tag{6.3.51}$$

for some $\mathbf{d} \in \mathcal{V}_{T,wx} \times \mathbb{Z}_q^n$ such that every co-ordinate of every $\mathcal{P}$ component belongs to $\{-1, 0, 1\}$ and the $\mathbb{Z}_q^n$ component is zero, under the $\mathsf{LWE}_{n,q,\chi}$ assumption for $\chi = D_{\mathbb{Z},r}$ and $q \geq e_T \cdot \lambda^{\omega(1)}$.

To simplify exposition, we denote by $OFF_{T,w} := \{-1, 0, 1\}^{m_w} \times \{0\}^n$, where m_w is the number of individual co-ordinates in a vector in $\mathcal{V}_{T,w}$. Clearly, $\mathbf{d} \in OFF_{T,wx}$. Note that this potential offset of ± 1 in each co-ordinate implies that this constrained PRF family is only *somewhat-homomorphic*, similar to [18] and Chapter 5.

Proof of Lemma 6.3.15. As in the proof of Lemma 6.3.14, we can claim the following to be true for *any* $w, x \in \{0,1\}^*$ such that $|wx| \leq |T|$, vector $\overline{\mathbf{k}} \in \overline{\mathcal{R}}_{T,|w|}$, and error $(\mathbf{e}_y) \in \mathcal{E}_{T,|w|}$:

$$\mathsf{PConstrain}_{T,w,x}(\mathbf{v}, \mathbf{s}) = \mathsf{Constrain}'_{T,w,x}(\mathbf{v}, \mathbf{s}, (\mathbf{e}_y)) \oplus \mathsf{Constrain}'_{T,w,x}(\mathbf{0}, \mathbf{0}, (-\mathbf{e}_y)) \ . \tag{6.3.53}$$

Also, as in the previous proof, we can assume that there are no "pass-throughs" in the execution of $\mathsf{PConstrain}_{T,w,x}$, because they will simply go through the $\overline{\mathsf{PConstrain}}_{T,w,x}$

evaluations on both sides of Equation (6.3.51) and will be passed as arguments to $\overline{\oplus}$.

By expanding $\overline{\mathsf{PConstrain}}_{T,w,x}(\cdot) = \lfloor \mathsf{PConstrain}_{T,w,x}(\cdot) \rfloor$ by Equation (6.3.30) and by Equation (6.3.53) above, we can expand the left hand side of Equation (6.3.51) as follows:

$$
\begin{aligned}
&\overline{\mathsf{PConstrain}}_{T,w,x}(\mathbf{v}_1, \mathbf{s}_1) \,\overline{\oplus}\, \overline{\mathsf{PConstrain}}_{T,w,x}(\mathbf{v}_2, \mathbf{s}_2) \\
&\quad = \Big\lfloor \mathsf{Constrain}'_{T,w,x}(\mathbf{v}_1, \mathbf{s}_1, (\mathbf{e}_y)_1) \oplus \mathsf{Constrain}'_{T,w,x}(\mathbf{0}, \mathbf{0}, (-\mathbf{e}_y)_1) \Big\rfloor \\
&\qquad \overline{\oplus}\, \Big\lfloor \mathsf{Constrain}'_{T,w,x}(\mathbf{v}_2, \mathbf{s}_2, (\mathbf{e}_y)_2) \oplus \mathsf{Constrain}'_{T,w,x}(\mathbf{0}, \mathbf{0}, (-\mathbf{e}_y)_2) \Big\rfloor \\
&\quad = \big(\mathsf{Constrain}'_{T,w,x}(\mathbf{v}_1, \mathbf{s}_1, (\mathbf{e}_y)_1) \oplus \mathsf{Constrain}'_{T,w,x}(\mathbf{0}, \mathbf{0}, (-\mathbf{e}_y)_1) \oplus \mathbf{d}_1 \big) \\
&\qquad \overline{\oplus}\, \big(\mathsf{Constrain}'_{T,w,x}(\mathbf{v}_2, \mathbf{s}_2, (\mathbf{e}_y)_2) \oplus \mathsf{Constrain}'_{T,w,x}(\mathbf{0}, \mathbf{0}, (-\mathbf{e}_y)_2) \oplus \mathbf{d}_2 \big) \\
&\quad = \Big\lfloor \big(\mathsf{Constrain}'_{T,w,x}(\mathbf{v}_1, \mathbf{s}_1, (\mathbf{e}_y)_1) \oplus \mathsf{Constrain}'_{T,w,x}(\mathbf{v}_2, \mathbf{s}_2, (\mathbf{e}_y)_2) \big) \\
&\qquad \oplus \big(\mathsf{Constrain}'_{T,w,x}(\mathbf{0}, \mathbf{0}, (-\mathbf{e}_y)_1) \oplus \mathsf{Constrain}'_{T,w,x}(\mathbf{0}, \mathbf{0}, (-\mathbf{e}_y)_2) \big) \oplus (\mathbf{d}_1 \oplus \mathbf{d}_2) \Big\rfloor \\
&\quad = \Big\lfloor \mathsf{Constrain}'_{T,w,x}\big((\mathbf{v}_1, \mathbf{s}_1, (\mathbf{e}_y)_1) \oplus (\mathbf{v}_2, \mathbf{s}_2, (\mathbf{e}_y)_2) \big) \\
&\qquad \oplus \mathsf{Constrain}'_{T,w,x}\big(\mathbf{0}, \mathbf{0}, (-\mathbf{e}_y)_1 + (-\mathbf{e}_y)_2 \big) \oplus (\mathbf{d}_1 \oplus \mathbf{d}_2) \Big\rfloor,
\end{aligned}
$$

for any $(\mathbf{e}_y)_1, (\mathbf{e}_y)_2 \in \mathcal{E}_{T,|w|}$ and some rounding offsets $\mathbf{d}_1, \mathbf{d}_2$, where the second equality follows from Fact 6.3.9, the third equality follows from the definition of $\overline{\oplus}$ and rearranging terms and the last by homomorphism of $\mathsf{Constrain}'$ (Lemma 6.3.3). If $(\mathbf{e}_y)_1, (\mathbf{e}_y)_2 \leftarrow \mathcal{W}_{T,|w|}$, we know by subgaussian additivity that $(\mathbf{e}_y)_1 + (\mathbf{e}_y)_2$ are subgaussian with parameter $\sqrt{2}r$. By an application of Lemma 6.3.10, we know that every $\mathcal{P}$ component (with associated subtree T') of $\mathsf{Constrain}_{T,w,x}(\mathbf{0}, \mathbf{0}, (-\mathbf{e}_y)_1 + (-\mathbf{e}_y)_2)$ is subgaussian with parameter $\sqrt{2}r \cdot \Phi_{T'}$ (and therefore every co-ordinate is bounded in magnitude by $R_{T'} = \sqrt{2}r \cdot \Phi_{T'} \cdot \omega(\sqrt{\log \lambda})$) and its $\mathbb{Z}_q^n$ component is zero with all but $\mathrm{negl}(\lambda)$ probability. We also have that every co-ordinate of every $\mathcal{P}$ component of $\mathbf{d}_1 \oplus \mathbf{d}_2$ is in $\{-2e_{T'}, \ldots, 0\}$, and its $\mathbb{Z}_q^n$ component is also zero.

We similarly use the definitions of $\overline{\mathsf{PConstrain}}_{T,w,x}$ and $\overline{\oplus}$ to expand the right

hand side of Equation (6.3.51) as follows:

$$
\begin{aligned}
&\overline{\mathsf{PConstrain}}_{T,w,x}\big((\mathbf{v}_1,\mathbf{s}_1)\ \overline{\oplus}\ (\mathbf{v}_2,\mathbf{s}_2)\big)\\
&\quad=\Big\lfloor \mathsf{PConstrain}_{T,w,x}\big(\big\lfloor(\mathbf{v}_1,\mathbf{s}_1)\oplus(\mathbf{v}_2,\mathbf{s}_2)\big\rfloor\big)\Big\rfloor\\
&\quad=\Big\lfloor \mathsf{PConstrain}_{T,w,x}\big((\mathbf{v}_1,\mathbf{s}_1)\oplus(\mathbf{v}_2,\mathbf{s}_2)\oplus\mathbf{d}_{12}\big)\Big\rfloor\\
&\quad=\Big\lfloor \mathsf{PConstrain}_{T,w,x}(\mathbf{v}_1,\mathbf{s}_1)\oplus\mathsf{PConstrain}_{T,w,x}(\mathbf{v}_2,\mathbf{s}_2)\oplus\mathsf{PConstrain}_{T,w,x}(\mathbf{d}_{12})\Big\rfloor\\
&\quad=\Big\lfloor \big(\mathsf{Constrain}'_{T,w,x}(\mathbf{v}_1,\mathbf{s}_1,(\mathbf{e}_y)_1)\oplus\mathsf{Constrain}'_{T,w,x}(\mathbf{0},\mathbf{0},(-\mathbf{e}_y)_1)\big)\\
&\qquad\oplus\big(\mathsf{Constrain}'_{T,w,x}(\mathbf{v}_2,\mathbf{s}_2,(\mathbf{e}_y)_2)\oplus\mathsf{Constrain}'_{T,w,x}(\mathbf{0},\mathbf{0},(-\mathbf{e}_y)_2)\big)\\
&\qquad\oplus\mathsf{PConstrain}_{T,w,x}(\mathbf{d}_{12})\Big\rfloor\\
&\quad=\Big\lfloor \big(\mathsf{Constrain}'_{T,w,x}(\mathbf{v}_1,\mathbf{s}_1,(\mathbf{e}_y)_1)\oplus\mathsf{Constrain}'_{T,w,x}(\mathbf{v}_2,\mathbf{s}_2,(\mathbf{e}_y)_2)\big)\\
&\qquad\oplus\big(\mathsf{Constrain}'_{T,w,x}(\mathbf{0},\mathbf{0},(-\mathbf{e}_y)_1)\oplus\mathsf{Constrain}'_{T,w,x}(\mathbf{0},\mathbf{0},(-\mathbf{e}_y)_2)\big)\\
&\qquad\oplus\mathsf{PConstrain}_{T,w,x}(\mathbf{d}_{12})\Big\rfloor\\
&\quad=\Big\lfloor \mathsf{Constrain}'_{T,w,x}\big((\mathbf{v}_1,\mathbf{s}_1,(\mathbf{e}_y)_1)\oplus(\mathbf{v}_2,\mathbf{s}_2,(\mathbf{e}_y)_2)\big)\\
&\qquad\oplus\mathsf{Constrain}'_{T,w,x}\big(\mathbf{0},\mathbf{0},(-\mathbf{e}_y)_1+(-\mathbf{e}_y)_2\big)\oplus\mathsf{PConstrain}_{T,w,x}(\mathbf{d}_{12})\Big\rfloor,
\end{aligned}
$$

for any $(\mathbf{e}_y)_1,(\mathbf{e}_y)_2\in\mathcal{E}_{T,|w|}$ and some rounding offset $\mathbf{d}_{12}$, where the second equality follows from Fact 6.3.9, the third and the last equalities follows from the homomorphism of $\mathsf{PConstrain}_{T,w,x}$ (Lemma 6.3.3), the fourth equality follows from Equation (6.3.53) above and the fifth follows by rearranging terms. As before, if $(\mathbf{e}_y)_1,(\mathbf{e}_y)_2\leftarrow\mathcal{W}_{T,|w|}$, we know that every $\mathcal{P}$ component of $\mathsf{Constrain}_{T,w,x}(\mathbf{0},\mathbf{0},(-\mathbf{e}_y)_1+(-\mathbf{e}_y)_2)$, with associated subtree T', is subgaussian with parameter $\sqrt{2}r\cdot\Phi_{T'}$ and its $\mathbb{Z}_q^n$ component is zero with all but $\mathrm{negl}(\lambda)$ probability. Also, since there are no pass-throughs, we have by the definition of $e_{T'}$ in Equation (6.3.23) (in particular, the fact that $e_{T'}\geq(e_{T'.l}\cdot(n\ell)+e_{T'.r})\cdot\lambda^{\omega(1)}$) that every $\mathcal{P}$ component of $\mathsf{PConstrain}_{T,wx,z}(\mathbf{d}_{12})$) is bounded in magnitude by some $C_{T'}$ in every co-ordinate such that $e_{T'}\geq C_{T'}\cdot\lambda^{\omega(1)}$, and its $\mathbb{Z}_q^n$ component is zero. It follows from these facts that every $\mathcal{P}$ component of $\mathsf{Constrain}'_{T,w,x}(\mathbf{0},\mathbf{0},(-\mathbf{e}_y)_1+(-\mathbf{e}_y)_2)\oplus\mathsf{PConstrain}_{T,w,x}(\mathbf{d}_{12}))$ is bounded in magnitude

in every co-ordinate by $B_{T'} := \sqrt{2}r \cdot \Phi_{T'} \cdot \omega(\sqrt{\log \lambda}) + C_{T'}$. Lastly, by Equation (6.3.34), we know that $e_{T'} \geq B_{T'} \cdot \lambda^{\omega(1)}$.

Based on what we have above, we get the following system of equations,

$$
\begin{aligned}
&\Pr_{\mathbf{s}_1,\mathbf{s}_2 \leftarrow \overline{\mathcal{D}}_{T,w}} [\overline{\mathsf{PConstrain}}_{T,w,x}(\mathbf{v}_1, \mathbf{s}_1) \mathbin{\overline{\oplus}} \overline{\mathsf{PConstrain}}_{T,w,x}(\mathbf{v}_2, \mathbf{s}_2) \\
&\qquad - \overline{\mathsf{PConstrain}}_{T,w,x}((\mathbf{v}_1, \mathbf{s}_1) \mathbin{\overline{\oplus}} (\mathbf{v}_2, \mathbf{s}_2)) \notin OFF_{T,wx}] \\
&= \Pr_{(\mathbf{s}_1,(\mathbf{e}_y)_1),(\mathbf{s}_2,(\mathbf{e}_y)_2)} \Big[\Big\lfloor \mathsf{Constrain}'_{T,w,x}\big((\mathbf{v}_1, \mathbf{s}_1, (\mathbf{e}_y)_1) \oplus (\mathbf{v}_2, \mathbf{s}_2, (\mathbf{e}_y)_2)\big) \\
&\qquad\qquad \oplus \mathsf{Constrain}'_{T,w,x}\big(\mathbf{0}, \mathbf{0}, (-\mathbf{e}_y)_1 + (-\mathbf{e}_y)_2\big) \oplus (\mathbf{d}_1 \oplus \mathbf{d}_2) \Big\rfloor \\
&\qquad - \Big\lfloor \mathsf{Constrain}'_{T,w,x}\big((\mathbf{v}_1, \mathbf{s}_1, (\mathbf{e}_y)_1) \oplus (\mathbf{v}_2, \mathbf{s}_2, (\mathbf{e}_y)_2)\big) \\
&\qquad\qquad \oplus \mathsf{Constrain}'_{T,w,x}\big(\mathbf{0}, \mathbf{0}, (-\mathbf{e}_y)_1 + (-\mathbf{e}_y)_2\big) \oplus \mathsf{PConstrain}_{T,w,x}(\mathbf{d}_{12}) \Big\rfloor \notin OFF_{T,wx} \Big] \\
&\leq \Pr_{(\mathbf{s}_1,(\mathbf{e}_y)_1),(\mathbf{s}_2,(\mathbf{e}_y)_2)} \Big[\mathsf{BAD}^1_{T,wx,B_{T'}}\big(\mathsf{Constrain}'_{T,w,x}\big((\mathbf{v}_1, \mathbf{s}_1, (\mathbf{e}_y)_1) \oplus (\mathbf{v}_2, \mathbf{s}_2, (\mathbf{e}_y)_2)\big)\big) \Big] \\
&\qquad + \Pr_{(\mathbf{s}_1,(\mathbf{e}_y)_1),(\mathbf{s}_2,(\mathbf{e}_y)_2)} \Big[\mathsf{BAD}^1_{T,wx,B_{T'}}\big(\mathsf{Constrain}'_{T,w,x}\big((\mathbf{v}_1, \mathbf{s}_1, (\mathbf{e}_y)_1) \oplus (\mathbf{v}_2, \mathbf{s}_2, (\mathbf{e}_y)_2)\big)\big) \Big] \\
&\qquad + \mathrm{negl}(\lambda),
\end{aligned}
$$

where the inequality above comes from the fact that if the $\mathsf{BAD}^1_{T,wx,B_{T'}}$ predicate is false for the value $\mathsf{Constrain}'_{T,w,x}((\mathbf{v}_1, \mathbf{s}_1, (\mathbf{e}_y)_1) \oplus (\mathbf{v}_2, \mathbf{s}_2, (\mathbf{e}_y)_2))$, then the right hand side of Equation (6.3.51) evaluates to $\big\lfloor \mathsf{Constrain}'_{T,w,x}\big((\mathbf{v}_1, \mathbf{s}_1, (\mathbf{e}_y)_1) \oplus (\mathbf{v}_2, \mathbf{s}_2, (\mathbf{e}_y)_2)\big) \big\rfloor$ with overwhelming probability, since the $\mathbb{Z}_q^n$ components of the additive terms are zero and the $\mathcal{P}$ components are bounded in magnitude by $B_{T'}$ in every co-ordinate with overwhelming probability. If in addition the $\mathsf{BAD}^2_{T,wx,R_{T'}+2e_{T'}}$ predicate on the same value is also false then we know that adding the extra terms does not push the $\mathcal{P}$ components of the main $\mathsf{Constrain}'$ term on the left hand side outside $\mathcal{P}$, and therefore produces no "carry" terms in the addition. Also, since $B_{T'} > R_{T'}$, we have that the only terms that could alter the rounding in the left hand side are the offset

$\mathbf{d}_1, \mathbf{d}_2$ terms, which combined, could only produce an offset of ± 1 in the rounding in the auxiliary part. Since the $\mathbb{Z}_q^n$ components of all additive terms are zero, and since there are no carry terms with high probability, we have that there is no difference in the $\mathbb{Z}_q^n$ components of the output, as needed.

Now, if we consider efficient adversaries, we can replace $\mathsf{Constrain}'_{T,w,x}(\mathbf{v}_b, \mathbf{s}_b, (\mathbf{e}_y)_b)$ (for $b \in \{1,2\}$) with a uniformly chosen $\mathbf{u}_b \leftarrow U(\mathcal{V}_{T,wx} \times \mathbb{Z}_q^n)$, whilst incurring only a negligible penalty, by Corollary 6.3.5, remembering the fact that each of the $\mathcal{P}$ components in the output are freshly generated without loss of generality and the fact that BAD^1 and BAD^2 are efficient to test. Lastly, note that $\mathbf{u} = \mathbf{u}_1 \oplus \mathbf{u}_2$ is also uniform. Therefore, for all efficient adversaries, we have the following

$$\begin{aligned}
&\Pr_{(\mathbf{s}_1,(\mathbf{e}_y)_1),(\mathbf{s}_2,(\mathbf{e}_y)_2)}\left[\mathsf{BAD}^1_{T,wx,B_{T'}}\left(\mathsf{Constrain}'_{T,w,x}((\mathbf{v}_1,\mathbf{s}_1,(\mathbf{e}_y)_1)\oplus(\mathbf{v}_2,\mathbf{s}_2,(\mathbf{e}_y)_2))\right)\right] \\
&\qquad \leq \Pr_{\mathbf{u}}[\mathsf{BAD}^1_{T,wx,B_{T'}}(\mathbf{u})] \leq \mathrm{negl}(\lambda), \\
&\Pr_{(\mathbf{s}_1,(\mathbf{e}_y)_1),(\mathbf{s}_2,(\mathbf{e}_y)_2)}\left[\mathsf{BAD}^2_{T,wx,R_{T'}+2e_{T'}}\left(\mathsf{Constrain}'_{T,w,x}((\mathbf{v}_1,\mathbf{s}_1,(\mathbf{e}_y)_1)\oplus(\mathbf{v}_2,\mathbf{s}_2,(\mathbf{e}_y)_2))\right)\right] \\
&\qquad \leq \Pr_{\mathbf{u}}[\mathsf{BAD}^2_{T,wx,R_{T'}+2e_{T'}}(\mathbf{u})] \leq \mathrm{negl}(\lambda),
\end{aligned}$$

which follow from the facts that $e_{T'} \geq B_{T'} \cdot \lambda^{\omega(1)}$, $q \geq e_{T'} \cdot \lambda^{\omega(1)}$ (and thus q is also superpolynomially bigger than $R_{T'} + 2e_{T'} < 3e_{T'}$) and Claim 6.3.12. □

CHAPTER VII

FAST PSEUDORANDOM FUNCTIONS IN PRACTICE

7.1 Overview

In this chapter, we give two new, optimized instantiations of the ring variant of the direct PRF construction, Construction 4.2.1 from Chapter 4, for parameters that offer high levels of concrete security against known classes of attacks, and provide very high-performance software implementations. We call this class of PRF candidates SPRING, which is short for "**s**ubset-**p**roduct with **r**ounding over a **ring**." We recall that these constructions are over some quotient ring R_p. We often identify $r \in R_p$ with the vector $\mathbf{r} \in \mathbb{Z}_p^n$ of its n coefficients in some canonical order. Let R_p^* denote the multiplicative group of units (invertible elements) in R_p. In all our instantiations in this chapter, we let n be a power of 2.

For a positive integer k, the SPRING family is the set of functions $F_{a,\vec{s}} : \{0,1\}^k \to \{0,1\}^m$ indexed by a unit $a \in R_p^*$ and a vector $\vec{s} = (s_1, \ldots, s_k) \in (R_p^*)^k$ of units. The function is defined as the "rounded subset-product"

$$F_{a,\vec{s}}(x_1, \ldots, x_k) := S\left(a \cdot \prod_{i=1}^{k} s_i^{x_i}\right),$$

where $S \colon R_p \to \{0,1\}^m$ for some $m \leq n$ is an appropriate "rounding" function. For example, we use the rounding function $\lfloor \cdot \rceil_2 \colon R_p \to R_2 \equiv \mathbb{Z}_2^n$ that maps each of its input's n coefficients to $\mathbb{Z}_2 = \{0,1\}$, depending on whether the coefficient is closer modulo p to 0 or to $p/2$. (Formally, each coefficient $b \in \mathbb{Z}_p$ is mapped to $\lfloor \frac{2}{p} \cdot b \rceil \in \mathbb{Z}_2$.)

We stress that the *known proof* of security (under ring-LWE) requires the modulus p to be very large, i.e., exponential in the input length k. Yet, as discussed in Section 4.3, the large modulus appears to be an artifact of the proof technique,

and the family appears not to *require* such large parameters for concrete security. Indeed, based on the state of the art in attacks on "noisy learning" problems like (ring-)LWE, it is reasonable to conjecture that the SPRING functions can be secure for rather small moduli p and appropriate rounding functions (see Section 7.4 for further details).

Because we aim to design practical functions, we instantiate the SPRING family with relatively small moduli p, rather than the large ones required by the theoretical security reductions from Section 4.2.3. This allows us to follow the same basic construction paradigm, while taking advantage of the fast integer arithmetic operations supported by modern processors. We instantiate the parameters as various (but not all) combinations of

$$n = 128, \quad p \in \{257, 514\}, \quad k \in \{64, 128\},$$

which (as explained below in Section 7.1.1) yields attractive performance, and allows for a comfortable margin of security. The choice of modulus $p \in \{257, 514\}$ is akin to the one made in SWIFFT, for a practical instantiation of a theoretically sound lattice-based collision-resistant hash function [61]. Also as in SWIFFT, our implementations build on Fast Fourier Transform-like algorithms modulo $q = 257$.

Working with small moduli p requires adjusting the rounding function $S(\cdot)$ in the SPRING construction so that its output on a uniformly random element of R_p^* does not have any noticeable bias (which otherwise would clearly render the function insecure as a PRF). We use rounding functions of the form $S(b) = G(\lfloor b \rceil_2)$, where $\lfloor \cdot \rceil_2 \colon R_p \to R_2$ is the usual coefficient-wise rounding function that provides (conjectured) indistinguishability from a potentially *biased* random function, and $G \colon R_2 \to \{0,1\}^m$ for some $m \leq n$ is an appropriate post-processing function that reduces or removes the bias. (In the theoretically secure constructions from the previous sections, G is effectively the identity function, because a huge modulus p ensures no noticeable bias in the rounded output.)

For each value of the modulus $p \in \{257, 514\}$ we have a different concrete instantiation, which we respectively call SPRING-BCH and SPRING-CRT. These instantiations differ mainly in the computation of the subset-products in R_p^*, and in the definition of the bias-reducing function G.

SPRING-BCH. In this instantiation, we use an odd modulus $p = q = 257$, which admits very fast subset-product computations in R_q^* using Fast Fourier Transform-type techniques (as mentioned above). However, because p is odd, the usual rounding function $\lfloor \cdot \rceil_2 \colon R_p \to R_2$ has bias $1/q$ on each of the output coefficients (bits). To reduce this bias, the function G multiplies the 128-dimensional, $1/q$-biased bit vector by the 64×128 generator matrix of a binary (extended) BCH error-correcting code with parameters $[n, m, d] = [128, 64, 22]$, yielding a syndrome with respect to the dual code. This simple and very fast "deterministic extraction" procedure (proposed in [4]) reduces the bias exponentially in the distance $d = 22$ of the code, and yields a 64-dimensional vector that is 2^{-145}-far from uniform when applied to a 128-dimensional bit vector of independent $1/q$-biased bits. However, this comes at the cost of outputting $m = 64$ bits instead of $n = 128$, as determined by the rate m/n of the code.

SPRING-CRT. In this instantiation, we use an even modulus $p = 2q = 514$, and decompose the subset-product computation over R_{2q}^* into its "Chinese remainder" components R_2^* and R_q^*. For the R_q^* component we use the same evaluation strategy as in SPRING-BCH, but for fast subset-products in the R_2^* component we need new techniques. We prove that the multiplicative group R_2^* decomposes into $n/2$ small cyclic groups, having power-of-two orders at most n. We also give explicit "sparse" generators for these cyclic components, and devise fast algorithms for converting between the "cyclic" representation (as a vector of exponents with respect to the generators) and the standard polynomial one. These tools allow us to transform a

subset-product in R_2^* into a subset-*sum* of vectors of (small) exponents with respect to the generators, followed by one fast conversion from the resulting vector of exponents to the polynomial it represents.

For rounding R_{2q}^*, we show that standard rounding of a uniformly random element of R_{2q}^* to R_2 directly yields $n-1$ independent and unbiased bits, so our function G simply outputs these bits. The main advantage over SPRING-BCH is the larger output size (almost twice as many bits), and hence larger throughput, and in the simpler and tighter analysis of the bias. On the other hand, we also show that the CRT decomposition of R_{2q}^* can be exploited somewhat in attacks, by effectively canceling out the R_2^* component and recognizing the bias of the rounded R_q^* component. Fortunately, for our parameters the best attacks of this type appear to take almost 2^{128} bit operations, and around 2^{119} space.

7.1.1 Implementations and Performance

We implement the two variants of SPRING described above, both for standalone evaluations on single inputs, and in a counter-like (CTR) mode that is able to amortize much of the work across consecutive evaluations. For the counter itself we use the Gray code, which is a simple way of ordering the strings in $\{0,1\}^k$ so that successive strings differ in only one position. Then when running SPRING in counter mode, each successive subset-product can be computed from the previous one with just one more multiplication by either a seed element or its inverse. More precisely, we store the currently computed subset-product $b := a\prod_{i=1}^{n} s_i^{x_i}$. (The Gray code starts with 0^k, so the initial subset-product is simply a.) If the next input x' flips the ith bit of x, then we update the old subset-product to $b' = b \cdot s_i$ if $x_i = 0$, otherwise $b' = b \cdot s_i^{-1}$.

For the SPRING-CRT instantiation, which works in $R_{2q}^* \cong R_2^* \times R_q^*$, we use two methods for computing (subset-)products in the R_2^* component. The first uses the cyclic decomposition of R_2^* as described above, and is the fastest method we have

found for computing a standalone subset-product "from scratch." The other method uses the native "carryless polynomial multiplication" (PCLMUL) instruction available in recent Intel processors, and/or precomputed tables, for single multiplications in the Gray code counter mode.

We benchmarked our implementations on a range of CPUs with several different microarchitectures. As a point of reference, we use the highly optimized AES benchmarks from eBACS [31], and the bitsliced implementation for Käsper and Schwabe [47]. We note that our implementations are in C (using compiler instrinsics for SIMD instructions); an optimized assembly implementation could probably run faster. We report our performances measures in Table 3, using high-end desktop processors (Core i7), and small embedded CPUs found in tablets and smart-phones (Atom and ARM Cortex). Even though the architectures of those machine are quite different, our results are very consistent: in counter mode, SPRING-BCH is between 8 and 10 times slower than AES (as measured by output throughput), while SPRING-CRT is about 4.5 times slower than AES (disregarding AES implementation with AES-NI when they are available). We expect similar results on other CPUs with similar SIMD engines. Finally, we mention that the very latest Intel CPUs (Haswell microarchitecture) include a new 256-bit wide SIMD engine with support for integer operations (AVX2). We expect that an AVX2 implementation of SPRING would run about twice as fast on those processors, yielding very compelling performance.

7.2 SPRING-BCH

Here we describe our first instantiation, SPRING-BCH, which works over R_q^* for a suitable prime q, and uses a BCH code for reducing the bias of the rounded subset-product.

Table 3: Implementation results for SPRING-BCH and SPRING-CRT with $n = 128$, in both standalone (SA) and Gray code counter mode (CTR). Speeds are presented in processor cycles per output byte, and are compared with the best known AES implementations.

	SPRING-BCH		SPRING-CRT		AES-CTR	
	SA	CTR	SA	CTR	w/o AES-NI	w/AES-NI
ARM Cortex A15	220	170	250	77	17.8	N/A
Atom	247	137	235	76	17	N/A
Core i7 Nehalem	74	60	76	29.5	6.9	N/A
Core i7 Ivy Bridge	60	46	62	23.5	5.4	1.3

7.2.1 Fast Subset Product in R_q

Efficient operations in the ring R_q were given in prior work by Lyubashevsky *et al.* [61] (following [68, 82, 60]). They give a Chinese Remainder decomposition of this ring as $R_q \cong \mathbb{Z}_q^n$, for prime $q = 1 \pmod{2n}$, and gave fast FFT-like algorithms for converting between (the standard polynomial representation of) R_q and $\mathbb{Z}_q^n$. In particular, the multiplicative group of units R_q^* is isomorphic to $(\mathbb{Z}_q^*)^n$. Since $\mathbb{Z}_q^*$ is cyclic and of order $q-1$, a subset-product in R_q reduces to a subset-sum of n-dimensional vectors of exponents modulo $q-1$ (with respect to some generator of $\mathbb{Z}_q^*$). Once the final vector of exponents have been computed, the corresponding element in $\mathbb{Z}_q^n$ can be computed by table lookups, and finally converted to its polynomial representation via the FFT-like algorithm from [61].

7.2.2 Rounding via BCH Code

Since q is odd, the usual rounding function $\lfloor \cdot \rceil_2 \colon R_q \to R_2$, when applied to a random input in R_q, outputs a ring element in R_2 whose (bit) coefficients are independent and have bias $1/q$. In this subsection we define a function $G \colon R_2 \to \{0,1\}^m$ that dramatically reduces this bias using a BCH code.

Definition 7.2.1 ([73]). *The* bias *of a distribution $X \in \{0,1\}^m$ with respect to*

$I \subseteq [m]$ is defined as

$$bias_I(X) = \left|\Pr\Big[\bigoplus_{i\in I} x_i = 0\Big] - \Pr\Big[\bigoplus_{i\in I} x_i = 1\Big]\right|.$$

Let max-bias(X) denote the maximal bias of X over all nonempty $I \subseteq [m]$.

Theorem 7.2.2 ([73]). *Let $X \in \{0,1\}^m$ be a random variable. Then*

$$2 \cdot \Delta(X, U_m) \leq \sqrt{2^m} \cdot \textit{max-bias}(X)$$

where $\Delta(X, U_m)$ denotes the statistical difference of X from the uniform distribution on m bits.

Proposition 7.2.3 ([4]). Let G be a generator matrix of a binary linear code with parameters $[n, m, d]$, and let $D \in \{0,1\}^n$ be a distribution of independent bits such that $\text{bias}_{\{i\}}(D) \leq \epsilon$ for every $i \in [n]$. Then $\text{max-bias}(G \cdot D) \leq \epsilon^d$.

From the above we get that when applied to a random input $b \in R_q$, the statistical distance of the distribution $S(b)$ from uniform is at most $(1/q)^d\sqrt{2^m}/2$. Note that in SPRING-BCH, we are actually applying G to $\lfloor b \rceil_2$ for a random *unit* $b \in R_q^*$, in which case the coefficients of $\lfloor b \rceil_2$ are not quite independent. Since we are anyway only heuristically modeling the subset-products as uniformly random and independent, we believe that it is safe to heuristically assume that G provides low bias in our instantiation.

In terms of implementation, generator matrices of BCH codes over $GF(2)$ are preferable, since the rows of the matrix are cyclic shifts of a single row, which facilitates fast implementation. We note that n is a power of 2, and any BCH code over $GF(2)$ is of length $2^t - 1$ for some integer t. To make the matrix compatible with an n that is a power of two, we use the extended-BCH code, which is obtained in a standard way by appending a parity bit to the codewords, and increases the code distance d by one. We finally note that for our chosen parameters $n = 128, m = 64$, the BCH

code with parameters $[127, 64, 21]$ and its extension with parameters $[128, 64, 22]$ have the largest known minimum distance for these specific rates.

7.3 SPRING-CRT

We now describe our second instantiation, called SPRING-CRT, which uses *unbiased* rounding on an *even* modulus of the form $p = 2q$, where q is an odd prime as in the instantiation from the previous section.

By the Chinese Remainder Theorem, the natural ring homomorphism $R_{2q} \to R_2 \times R_q$ is a ring isomorphism, and moreover, there is an explicit map which lets us convert back and forth between the two representations. Specifically, it is easy to verify that the pair $(b_2, b_q) \in R_2 \times R_q$ corresponds to

$$b = q \cdot \overline{b}_2 + (q+1) \cdot \overline{b}_q \pmod{2q} \tag{7.3.2}$$

for arbitrary $\overline{b}_2, \overline{b}_q \in R_{2q}$ such that $\overline{b}_2 = b_2 \pmod 2$ and $\overline{b}_q = b_q \pmod q$. The CRT isomorphism also induces a group isomorphism between R_{2q}^* and $R_2^* \times R_q^*$, and thus lets us represent the seed elements and their subset-products as pairs in $R_2^* \times R_q^*$. We compute products in the R_q^* component as detailed in Section 7.2.1 above. In the following subsections, we define an unbiased rounding function from R_{2q}^* to R_2, and give fast algorithms for computing products in the R_2^* component.

7.3.1 Unbiased Rounding of R_{2q}^*

We start by describing how the rounding function from R_{2q} to R_2 can be computed directly from the Chinese remainder components $(b_2, b_q) \in R_2 \times R_q$ of a given $b \in R_{2q}$. As above, let $\overline{b}_q, \overline{b}_2 \in R_{2q}$ denote arbitrary mod-$2q$ representatives of b_2, b_q. By Equation (7.3.2) and the definition of the rounding function $\lfloor \cdot \rceil \colon R_{2q} \to R_2$,

$$\lfloor b \rceil_2 = \left\lfloor q(\overline{b}_q + \overline{b}_2) + \overline{b}_q \right\rceil_2 = \left\lfloor (\overline{b}_q + \overline{b}_2) + \overline{b}_q / q \right\rceil.$$

If we choose the coefficients of $\overline{b}_q$ from $[-q/2, q/2) \cap \mathbb{Z}$, then each coefficient of $\overline{b}_q/q$ is in the interval $[-1/2, 1/2)$, so

$$\lfloor b \rceil_2 = \overline{b}_q + \overline{b}_2 \bmod 2. \tag{7.3.4}$$

Equivalently, the coefficient vector of $\lfloor b \rceil_2$ is the exclusive-or of the coefficient vector of $\overline{b}_2$ and the least-significant bits of the coefficients of $\overline{b}_q$.

In SPRING-CRT, we need an unbiased rounding function S from the *unit group* $R_{2q}^* \cong R_2^* \times R_q^*$ to R_2. An element of R_2, viewed as a polynomial, is a unit if and only if the sum of its coefficients is odd. So for a uniformly random element of R_2^*, any fixed choice of $n-1$ coefficients (e.g., all but the constant term) are uniformly random and independent, and the remaining one is determined. Because of Equation (7.3.4) above, any fixed choice of $n-1$ coefficients of $\overline{b}_2$ are uniformly random and independent, over the random choice of $b_2 \in R_2^*$ alone. Therefore, we define our generalized rounding function on $b \in R_{2q}^*$ to output a fixed $n-1$ bits of $\lfloor b \rceil_2 \in R_2$, which is perfectly unbiased.

Note that the above argument depends only on the random choice of the R_2^* component, and doesn't use any of the randomness in the R_q^* component. Using such an argument, $n-1$ independent and unbiased bits is the most we can possibly obtain. Since the number of units in R_{2q}^* is exactly $(q-1)^n \cdot 2^{n-1}$, which is divisible by 2^n, it seems plausible that there could exist a rounding function that outputs n (nearly) unbiased bits given a random unit in R_{2q}^*, but so far we have not been able to find such a function. The main difficulty seems to be that the coefficients of the representative $\overline{b}_q$ are noticeably biased modulo 2.

7.3.2 Fast Arithmetic in R_2^*

We now give an algebraic decomposition of the group R_2^*, and present fast algorithms for performing subset-products and associated arithmetic operations.

The following theorem says that the unit group R_2^* decomposes into the product

of several small cyclic components, having power-of-2 orders at most n. The full proof of this theorem appears in Section 7.7.

Theorem 7.3.1 (Combining Lemmas 7.7.4 and 7.7.5). *Define* $g_{0,0} = 1+(1+x)$ *and* $g_{i,k} = 1 + (1+x)^{2^i+k}$ *for* $1 \leq i < \lg(n)$ *and* odd $k \in \{1, \ldots, 2^i\}$. *Then*

$$R_2^* \cong C_2^{n/4} \times C_4^{n/8} \times \ldots \times C_{n/2}^1 \times C_n^1 = \prod_{i=1}^{\lg(n)-1} C_{2^i}^{2^{j-i-1}} \times C_n,$$

with each $g_{i,k}$ *being a generator of one of the* $C_{n/2^i}$ *cyclic components.*

There are several ways of representing elements in R_2^*, which each allow for certain arithmetic operations to be performed more or less efficiently. We use the following three representations, the first of which is very good for fast multiplication, and the last of which is used for rounding. (As we shall see, the middle one is a convenient intermediate representation.)

1. Using the cyclic decomposition given in Theorem 7.3.1, we can represent an element by its tuple of integer exponents with respect to the generators $g_{i,k}$. We call this the *exponent representation*.

2. We can represent elements in R_2 by their vectors of $\mathbb{Z}_2$-coefficients with respect to what we call the *radix basis* $\{(1+x)^i\}_{0 \leq i < n}$. (An element is in R_2^* if and only if its coefficient for the basis element $(1+x)^0 = 1$ is 1.) The name of this basis arises from the fact that $(1+x)^n = 1 + x^n = 0 \pmod 2$, and therefore the coefficients can be thought of as digits in the "radix" $1 + x$.

3. Finally, elements in R_2 can be represented by their vectors of $\mathbb{Z}_2$-coefficients with respect to the *power basis* $\{x^i\}_{0 \leq i < n}$, i.e., in the usual way as polynomials in x.

We now give algorithms for efficiently converting from the exponent representation to the power representation, using the radix representation as an intermediary.

From exponents to radix basis. We first make a few useful observations about the radix basis, and how powers of the generators $g_{i,k}$ look in this basis.

1. In the radix basis, multiplication by an element of the form $(1+x)^j$ corresponds to shifting the input's coefficient vector j places (and discarding the "top" j coefficients), since $(1+x)^n = 0$ in R_2. Therefore, multiplication by $1+(1+x)^j$ corresponds to taking the exclusive-or of the input's coefficient vector with that vector shifted by j positions.

2. For any j and ℓ, we have that $(1+(1+x)^j)^{2^\ell} = 1+(1+x)^{j\cdot 2^\ell} \in R_2^*$, since the intermediate binomial coefficients $\binom{2^\ell}{i}$ for $0 < i < 2^\ell$ are all even.

3. Raising any generator $g_{i,k}$ to half its order yields $g_{i,k}^{n/2^{i+1}} = 1+(1+x)^j$, where $j = n/2 + (n/2^{i+1})k$. Moreover, the product of any two elements of this type, for $n/2 \le j_1, j_2 < n$, is

$$(1+(1+x)^{j_1})(1+(1+x)^{j_2}) = 1+(1+x)^{j_1}+(1+x)^{j_2}.$$

   Thus, a subset-product of elements of this type can be computed as $\prod_{j\in\mathcal{I}}(1+(1+x)^j) = 1+\sum_{j\in\mathcal{I}}(1+x)^j$, for any $\mathcal{I} \subseteq \{n/2,\ldots,n-1\}$.

Now let the exponent representation of some $b \in R_2^*$ be $\{e_{i,k}\}$, where each $e_{i,k}$ denotes the exponent of the generator $g_{i,k}$. Write $e_{i,k} = \sum_{\ell=0}^{\lg(n)-i-1} e_{i,k,\ell}\cdot 2^\ell$, i.e., each $e_{i,k,\ell}$ is the ℓth bit of $e_{i,k}$, and observe that

$$g_{i,k}^{e_{i,k}} = \prod_{\ell=0}^{\lg(n)-i-1} \left(g_{i,k}^{2^\ell}\right)^{e_{i,k,\ell}}, \tag{7.3.7}$$

where we know by Item 2 above that $g_{i,k}^{2^\ell} = 1+(1+x)^{(2^i+k)\cdot 2^\ell}$.

We can now describe the algorithm that converts from exponent to radix representation. We effectively decompose the given powers $e_{i,k}$ of $g_{i,k}$ according to Equation (7.3.7), which we can then compute by Items 1 and 2 above. We note that Item 3 lets us handle all the most significant bits of all the exponents very quickly in one

shot. (This yields a practical but not asymptotic improvement over handling these bits more naively.) The precise details are given in Algorithm 3 below.

Algorithm 3 Algorithm to convert from exponent to radix representation

Input: Exponents $e_{i,k} \in [0, n/2^i)$ for positive odd $k < 2^i$ when $0 < i < \lg(n)$, and $k = 0$ when $i = 0$. ▷ Let $e_{i,k,\ell}$ denote the ℓth bit of $e_{i,k}$.
Output: A bit vector $\mathbf{b} \in \mathbb{Z}_2^n$ representing the coefficients of b in the radix basis.
$\mathbf{b} \leftarrow (1, 0, \ldots, 0)$ ▷ Initialize the vector $\mathbf{b}$ to represent $1 \in R_2^*$
for every valid (i, k) pair **do**
    if $e_{i,k,\lg(n)-i-1} = 1$ **then**
        $b[n/2 + k \cdot n/2^{i+1}] \leftarrow 1$
    end if
end for
for every valid (i, k) pair **do**
    for $\ell = (\lg(n) - 2) - i$ down to 0 **do**
        if $e_{i,k,\ell} = 1$ **then**
            $\mathbf{b} \leftarrow \mathbf{b} \oplus (\mathbf{b} \gg ((2^i + k) \cdot 2^\ell))$ ▷ The shift-and-XOR operation.
        end if
    end for
end for

If the length of the coefficient vector is considered to be the word-size, then apart from the most significant bits of the exponents (which are handled in one word operation in total), the other bits are handled in one shift-and-XOR operation each, which is a constant number of word operations each. Since the exponents take $n - 1$ bits in total, the algorithm performs a total of $O(n)$ word operations. (Since each word is n bits long, the bit complexity of Algorithm 3 is $O(n^2)$.)

From radix basis to power basis. For the second step, we have a bit vector $\mathbf{b} \in \mathbb{Z}_2^n$ representing some $b \in R_2$ with respect to the radix basis, and wish to convert

to the power basis. We express b as follows:

$$\begin{aligned} b &= \sum_{i=0}^{n-1} b_i(1+x)^i = \sum_{i=0}^{n/2-1} b_i(1+x)^i + (1+x)^{n/2} \sum_{i=0}^{n/2-1} b_{i+n/2}(1+x)^i \\ &= \sum_{i=0}^{n/2-1} b_i(1+x)^i + (1+x^{n/2}) \sum_{i=0}^{n/2-1} b_{i+n/2}(1+x)^i \qquad (7.3.8) \\ &= \left(\sum_{i=0}^{n/2-1} (b_i + b_{i+n/2})(1+x)^i \right) + x^{n/2} \sum_{i=0}^{n/2-1} b_{i+n/2}(1+x)^i \pmod 2, \end{aligned}$$

where (7.3.8) follows from the fact that $(1+x)^{2^j} = 1 + x^{2^j} \pmod 2$ (since $\binom{2^j}{i}$ is even for every $0 < i < 2^j$), and n is a power of 2. Converting the n-bit vector $\mathbf{b}$ therefore reduces to two conversions of $n/2$-bit vectors, namely, the top half of $\mathbf{b}$ and the exclusive-or of the top and bottom halves of $\mathbf{b}$. This directly yields a simple divide-and-conquer algorithm to transform the coefficient vector $\mathbf{b}$ in place, which is detailed in Algorithm 4 below. The number of bit operations follows the simple recursive equation $T(n) = 2T(n/2) + n/2$, which solves to $T(n) = O(n \log n)$.

Algorithm 4 Algorithm to transform from radix basis to power basis of R_2

```
1: procedure RADIX-TO-POWER(b, f, ℓ)
Require: array b, index f and length ℓ    ▷ The initial call is made with f = 0 and
    ℓ = n
Ensure: subvector b[f, f + ℓ − 1] converted to power basis
2:     if ℓ > 1 then
3:         for i = 0 to ℓ/2 − 1 do
4:             b[f + i] ← b[f + i] ⊕ b[f + ℓ/2 + i]
5:         end for
6:         RADIX-TO-POWER(b, f, ℓ/2)
7:         RADIX-TO-POWER(b, f + ℓ/2, ℓ/2)
8:     end if
9: end procedure
```

7.4 Security Analysis

In this section we analyze the security of our construction against known classes of attacks, and introduce new attacks specific to the structure of R_{2q}.

7.4.1 Overview of Known Attacks

The concrete security of the SPRING PRF for practical parameters is not well understood, but to date there are no known attacks that nontrivially exploit the internal subset-product structure. As discussed earlier, the SPRING construction follows the paradigm from Chapter 4 which results in a PRF that is secure against all efficient adversaries, assuming the hardness of the (ring-)LWE problem (appropriately parameterized). Informally, the ring-LWE problem asks the adversary to distinguish many pairs $(a_i, b_i) \in R_p \times R_p$, where each a_i is chosen uniformly and $b_i \approx a_i \cdot s$ is its *noisy* product with a secret ring element s, from uniformly random pairs. The security reductions make two assumptions that do not hold in our instantiations: (1) the parameter p is exponential in the input length k of the PRF, and (2) the seed elements s_i are all "small" ring elements in R; more precisely, they are drawn from the error distribution from the underlying ring-LWE assumption. However, as we shall see in what follows, relaxing these requirements do not appear to introduce any concrete attacks against the function family.

For the sake of modeling certain attacks against SPRING, we can think of it as a LWE-type learning problem. The difference here is that *all* ring elements output by SPRING have rounding errors in them, whereas ring-LWE releases the multiplicand a without any error. In this respect, attacking SPRING seems potentially harder than attacking ring-LWE.

The main classes of attacks against noisy learning problems akin to LWE are: (1) brute-force attacks on the secret, (2) combinatorial-type attacks following [15, 95, 71], (3) lattice reduction attacks, and (4) algebraic attacks following [7]. We consider each of these in turn. We note that the lattice and algebraic attack strategies described below apply to (ring-)LWE with our parameters. It is not clear whether these attacks will adapt to SPRING, where multiplicands are not known exactly, but to be conservative we assume that they might. While most of these attacks

take a prohibitively large amount of time and/or space (more than 2^{200}), one kind of birthday-type attack technique performs reasonably well against SPRING-CRT. Even in this case, its running time is nearly 2^{128} bit operations and its space requirements are about 2^{119}, when $n = 128$.

Brute-force and combinatorial attacks. A brute-force attack involves searching for a secret $s_i \in R_p^*$, or for the round-off terms in enough samples to uniquely determine an s_i. The secret and round-off terms come from sets of size at least $(p/2)^n$, which is prohibitively large for all our parameters. Combinatorial (or "generalized birthday") attacks on noisy learning problems [15, 95] work by drawing an huge number of samples and finding (via birthday collisions) small combinations that sum to lie in a small enough subgroup, then testing whether the noise can be detected. This works for small error rates because the small combinations still retain small error terms. In the case of SPRING-CRT, this style of attack looks the most promising, and a concrete attack in this vein is developed further in Section 7.4.2.

Lattice attacks. Lattice attacks on (ring-)LWE typically work by casting it as a bounded-distance decoding (BDD) problem on a certain class of random lattices (see for instance [71, 55, 57, 92]). At a high level, the attack draws a sufficiently large number L of samples $(a_i, b_i) \in R_p \times R_p$, so that the secret (in the LWE case) is uniquely determined with good probability. With error rate $1/2$, we need $L \geq \lg(p/2)$ by a simple information-theoretic argument. The attack collects the samples into vectors $\vec{a}, \vec{b} \in R_p^L$, and considers the "$p$-ary" lattice $\mathcal{L}$ of dimension $N = nL$ (over $\mathbb{Z}$) corresponding to the set of vectors $s \cdot \vec{a} \in R_p^L$ for all $s \in R_p$. It then attempts to determine whether $\vec{b}$ is sufficiently close to $\mathcal{L}$, which corresponds to whether (a_i, b_i) are LWE samples or uniform. In our setting, because the error rate $1/2$ is so large, the distance from $\vec{b}$ to $\mathcal{L}$ (in the LWE case) is nearly the minimum distance of the lattice, up to a constant factor no larger than four (this is a conservative bound). Therefore,

for the attack to succeed it needs to solve BDD (or the shortest vector problem SVP) on $\mathcal{L}$ to within an very small constant approximation factor. For the parameters in our instantiations, the lattice dimension is at least $N \geq n \lg(p/2) \geq 896$ (and likely more). For this setting, the state of the art in BDD and SVP algorithms [29, 57, 72], take time at least $2^{0.48N} \geq 2^{430}$, and likely more. Moreover, the SVP algorithm of [72], which appears to provide the best heuristic runtime in this setting, as a most conservative estimate requires space at least $2^{0.18N} \geq 2^{160}$.

Algebraic attacks. Finally, the algebraic "linearization" attack of Arora and Ge [7] yields a lower bound on p for security. The attack is applicable when every coefficient of every error term is guaranteed to belong to a known set of size d; in our setting, $d = p/2$. The attack requires at least N/n ring-LWE samples to set up and solve a dense linear system of dimension N, where

$$N = \binom{n+d}{n} \approx 2^{(n+d)\cdot H(n/(n+d))}$$

and $H(\delta) = -\delta \lg(\delta) - (1-\delta)\lg(1-\delta)$ is the binary entropy function for $\delta \in (0,1)$. Therefore, the attack requires time and space at least N^2, which is at least 2^{384} for even the most aggressive of all our parameters.

7.4.2 Birthday-type Attack on SPRING-CRT

We now describe a specific birthday-type attack on SPRING-CRT, which exploits the structure of the ring R_{2q}. The main idea is to cancel out the R_2 component and to detect the bias in the remaining R_q component.

To do this, we first split the input x into two parts, as $x = y \| z$ for y, z of certain lengths. Then the SPRING-CRT function (for simplicity, without dropping a bit after rounding) can be written as

$$F(y\|z) = \lfloor a \cdot S_y \cdot S_z \rceil_2,$$

where S_y and S_z respectively represent the subset product of the keys s_i indicated by the bits of y and z.

The basic goal in the attack is to try to find values y and y' such that $S_y = S_{y'} \bmod 2$. If we have two such values, then $a \cdot S_y \cdot S_z = a \cdot S_{y'} \cdot S_z \bmod 2$ for any z, so the R_2 component of the output will be the same for the inputs $y\|z$ and $y'\|z$. By Equation (7.3.4) in Section 7.3.1, this implies that in $F(y\|z) \oplus F(y'\|z)$, the respective R_2 components cancel each other out. Since rounding of a uniform element in R_q has a bias of $1/q$ in each coefficient, the bits of $F(y\|z) \oplus F(y'\|z)$ will be the sum of two biased bits, i.e., the bias is $1/q^2$. This can be detected using $q^4/4$ pairs of output bits (with varying z).

If we repeat the test for 2^n different choices of (y, y'), with high probability, one choice satisfies $S_y + S_{y'} = 0 \bmod 2$, and we would be able to detect the bias by the method detailed above. (By contrast, the test would not detect such bias in a truly random function, with high probability.) We can collect the data for the attack with $2^{n/2}$ distinct choices of y and y', each of them using $q^4/(4n)$ values of z. This requires $2^{n/2} \cdot q^4/(4n)$ queries *and* space, and a time complexity of $2^n \cdot q^4/2$.

Generalizing the attack. We can generalize this analysis using y and y' such that $(S_y + S_{y'})^2 = 0 \bmod 2$. This implies that the $S_{y,2} + S_{y',2}$ is a multiple of $x^{n/2} + 1$, where $S_{y,2} = S_y \bmod 2$ and similarly for $S_{y',2}$. Thus, c_i and $c_{i+n/2}$, the coefficients of x^i and $x^{n+i/2}$ respectively in $S_{y,2} + S_{y',2}$, are the same. This implies that if we XOR the lower and upper halves of $F(y\|z) \oplus F(y'\|z)$, we can effectively remove the R_2 component as above, and then can recognize the bias in the R_q component. Since we sum four bits to remove the R_2 component, we reduce the bias, but a random pair y, y' satisfies the condition with probability $2^{-n/2}$ instead of 2^{-n}. This gives an attack with query and space complexity $2^{n/4} \cdot q^8/(4n)$ and time complexity $2^{n/2} \cdot q^8/4$. This can be generalized further: if we use y and y' such that $(S_y + S_{y'})^t = 0 \bmod 2$ (for t

a power of 2), we reach a time complexity of $2^{n/t} \cdot q^{4t}/(2t)$.

With $q = 257$ and $n = 128$, our best attack on SPRING-CRT (using $t = 2$) has time complexity roughly $2^{64+64-2} = 2^{126}$, and query and space complexity roughly $2^{n/2} \cdot q^8/(4n) \approx 2^{119}$.

7.5 Implementation Details

The SPRING design is targeted for efficient implementation using SIMD instructions, and a well-optimized implemention allows us to reach throughputs that are not too far from those of classical symmetric-key constructions.

SIMD instructions perform a given operation on multiple data in parallel. Processors with a SIMD engine usually come with a set of dedicated registers, which can contain a vector of integers or floating point data, and the SIMD instruction set computes arithmetic operations in parallel on the vectors elements, e.g., addition, multiplication, bitwise operations, rotations, etc. as well as some permutations of the vector elements. SIMD instructions were introduced in personal computers to improve the efficiency of multimedia computations, and are now very widely available. SIMD engines with 128-bit wide vectors are available on all desktop processors (SSE2 on x86/x86_64, Altivec on PowerPC, NEON on ARM), and even on embedded platforms such as smart-phones and tablets, with ARM Cortex-A or Intel Atom. Very recently, Intel has introduced AVX2, with integer operations on 256-bit SIMD vectors.

We implemented the two instantiations of SPRING defined in Sections 7.2 and 7.3. SPRING-BCH involves a subset-product in R_q^*, followed by rounding and bias reduction (using the BCH code), while SPRING-CRT involves a subset product in R_{2q}^* followed by rounding. As described in Section 7.3.1, this can be implemented as separate subset-products in R_2^* and R_q^*, followed by an extraction of the least significant bits in the R_q^* component and an exclusive-or with the R_2^* component. For each version, we have an implementation of the PRF with standalone subset-products, and

an amortized implementation in Gray code counter mode where we just perform one ring multiplication before each rounding operation. In the following subsections we explain how to efficiently implement the main operations.

7.5.1 Computations in R_2^*

Subset-sum and conversion from exponent to power basis. We use the cyclic decomposition of R_2^* given in Theorem 7.3.1, and store the key in exponent representation, so that the subset-product is a subset-sum of the exponents. A polynomial in R_2 (of degree less than 128) is represented by 32 one-bit indices, 16 two-bit indices, 8 three-bit indices, 4 four-bit indices, 2 five-bit indices, 1 six-bit index, and 1 seven-bit index. We store all 64 indices as 8-bit integers, and use SIMD instructions to compute the sum. Since all the cyclic groups have an order that is a power of 2, we can use 8-bit additions, and remove the extra bits at the end. The conversion to the power basis is done using Algorithms 3 and 4. Algorithm 4 is rewritten iteratively using shift, mask and xor instructions, taking advantage of the inherent parallelism of bitwise operations, as shown in Algorithm 5.

Algorithm 5 Iterative version of Algorithm 4 using the parallelism of bitwise operations

1: **procedure** RADIX-TO-POWER($\mathbf{b}$)
Require: 128-bit vector $\mathbf{b}$
2: $\mathbf{b} \leftarrow \mathbf{b} \oplus (\mathbf{b} \wedge \texttt{0xffffffffffffffff0000000000000000}) \gg 64$
3: $\mathbf{b} \leftarrow \mathbf{b} \oplus (\mathbf{b} \wedge \texttt{0xffffffff00000000ffffffff00000000}) \gg 32$
4: $\mathbf{b} \leftarrow \mathbf{b} \oplus (\mathbf{b} \wedge \texttt{0xffff0000ffff0000ffff0000ffff0000}) \gg 16$
5: $\mathbf{b} \leftarrow \mathbf{b} \oplus (\mathbf{b} \wedge \texttt{0xff00ff00ff00ff00ff00ff00ff00ff00}) \gg 8$
6: $\mathbf{b} \leftarrow \mathbf{b} \oplus (\mathbf{b} \wedge \texttt{0xf0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0}) \gg 4$
7: $\mathbf{b} \leftarrow \mathbf{b} \oplus (\mathbf{b} \wedge \texttt{0xcccccccccccccccccccccccccccccccc}) \gg 2$
8: $\mathbf{b} \leftarrow \mathbf{b} \oplus (\mathbf{b} \wedge \texttt{0xaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa}) \gg 1$
9: **end procedure**

Polynomial multiplication. In counter mode, we found it more efficient to compute a single ring multiplication directly than to use the exponent representation.

On recent Intel CPUs (starting from the Westemere architecture introduced in 2010) and AMD CPUs (starting from the Bulldozer architecture introduced in 2011), there is a carry-less multiplication operation, `pclmulqdq`, that computes a 64-bit polynomial multiplication modulo two. This gives a very efficient implementation of the R_2 multiplication.

Alternatively, we take take advantage of the fact that one of the operands is always a polynomial from the key (or its inverse). Therefore, we can see it as a multiplication by a fixed element in R_2, which is a linear operation. We can precompute tables corresponding to this linear operation with 8-bit subsets of the input range, and compute the full multiplication using $n/8$ table accesses and xors.

More precisely, we precompute $z \cdot s$ for all polynomials z of degree less than 8, and we write a degree-128 polynomial z as $z_0 + x^8 \cdot z_1 + \cdots + x^{120} \cdot z_{15}$, where all the z_i are of degree at most 7. Then we can compute $z \cdot s$ as $(z_0 \cdot s) + x^8 \cdot (z_1 \cdot s) + \cdots + x^{120} \cdot (z_{15} \cdot s)$, which requires only 16 table accesses, rotations, and xors. This trick takes about 1MB of extra memory to store the tables, but this is negligible on the platforms we target.

7.5.2 Computations in R^*_{257}

Following [61], we use the Chinese Remainder Theorem isomorphism of the ring $R_q \cong \mathbb{Z}_q^n$ when $q = 1 \pmod{2n}$ is prime. A product in R_q therefore corresponds to a component-wise multiplication of vectors in $\mathbb{Z}_q^n$. Moreover, there are fast FFT-like algorithms, often called "number theoretic transforms" (NTT), for converting between the polynomial representation of R_q and the n-fold product ring $\mathbb{Z}_q^n$.

Subset-sum and conversion. Since the ring elements we multiply are all part of the key, we can generate and store them as vectors in the product ring $\mathbb{Z}_q^n$. Moreover, since these elements are all unit, their entries in $\mathbb{Z}_q^n$ are non-zero, and we can actually store the discrete logarithms of the entries (with respect to some generator of $\mathbb{Z}_q^*$), so that the subset-product becomes a subset-sum.

Exponentiation by the final summed exponents can be implemented with a simple table lookup. However, we found a slightly more efficient version using vector permutation (`pshufb` in SSSE3) instructions as a 4-bit to 8-bit parallel table lookup. We use the fact that $g^{a+16\times b} = g^a \cdot (g^{16})^b$, where a and b are both 4-bit values, and we use 4-bit to 8-bit tables for g^x and $(g^{16})^x$.

Product and conversion. In Gray code counter mode, we do not use the exponent representation, because a point-wise multiplication is more efficient than a point-wise addition followed by exponentiations. This is because the point-wise multiplication can be parallelized easily while the exponentiation requires either serial table lookups, or a more complex sequence of SIMD operations.

7.5.2.1 NTT

The bottleneck of our function is the NTT computation, therefore we have to optimize this part aggressively. In our implementation, we reuse the code from the SIMD hash function [52] which happens to use the same parameters as the transformation needed in SPRING. The main tricks used in this implementation are:

Representation of elements. Element in $\mathbb{Z}_{257}$ are stored as signed 16-bit words. The choice of the modulus 257 allows an efficient implementation of the field operations, because 257 is a prime and $256 = -1 \pmod{257}$. Moreover, $\mathbb{Z}_{257}^*$ is a cyclic group of 256 elements, where the subset sum of the logarithms can be computed with a simple 8-bit addition.

Reduction. We use `(x&255) - (x>>8)` to do a partial reduction modulo 257, with the output in $[-127, 383]$. When a full reduction to a smaller range is needed, we subtract 257 to values greater than 128 to reduce the range to $[-128, 128]$. This can be performed completely with SIMD instructions and does not require any division.

We note that it is not necessary to perform a reduction after each field operation, because we have some extra bits in a 16-bit word; we have to study the NTT algorithm to find out where reductions are needed.

Multiplication. To compute a multiplication in $\mathbb{Z}_{257}$, we reduce both operands to $[-128, 128]$, and the result can be computed with a single 16-bit multiplication without any overflow.

Using a two-dimensional NTT. Because SIMD instructions compute the same operation on each element of the vectors, we do not use the classical radix-2 NTT algorithm. Instead, we rewrite the one-dimensional NTT as a two-dimensional one. In our implementation, we rewrite an NTT of size 64 as a two-dimensional NTT of size 8×8. The input data is seen as a 8×8 matrix, and the computation of the NTT_{64} is done in three steps:

- First we compute 8 parallel NTT_8 on the columns of the matrix using a decimation in time algorithm.
- We multiply by the twiddle factors, transpose the matrix, and permute the row and the columns following the bit reversal order.
- Then we compute 8 parallel NTT_8 on the columns of the matrix using a decimation in frequency algorithm.

The first and the last step are easy to parallelize with SIMD instructions because they compute the same transformation on 8 independent inputs. Moreover, the root of unity used in the NTT_8 is 4, so the multiplications needed for the NTT_8 are simply bit shifts. The transposition can be implemented using merge operations available on most SIMD instruction sets (e.g., `punpcklwd`/`punpckhwd` in SSE).

For the 128-dimensional NTT, we reused the code of the NTT_{64}, and we have to perform an extra layer of butterfly operations and multiplications by twiddle factors (we decompose the NTT_{128} as an NTT_{64} and a NTT_2).

7.5.3 Reducing Bias with a BCH Code

After rounding the R_{257} computation output, we are left with a n-dimensional vector over $\mathbb{Z}_2$, each element with a bias of $1/257$. We apply the generator polynomial of a BCH code in order to reduce the output's bias. Specifically, for the case of $n = 128$ we apply the extension using a parity bit on the BCH code with parameters $[127, 64, 21]$ in order to gain a generator for a code with distance 22. Therefore, the whole computation is done using just a few shift and xor instructions, and one final negate instruction. We use the BCH generator polynomial

$$1 + x^2 + x^7 + x^8 + x^{10} + x^{12} + x^{14} + x^{15} + x^{16} + x^{23} + x^{25} + x^{27} + x^{28} + x^{30} + x^{31}$$
$$+ x^{32} + x^{33} + x^{37} + x^{38} + x^{39} + x^{40} + x^{41} + x^{42} + x^{44} + x^{45} + x^{48} + x^{58} + x^{61} + x^{63}$$

since it matches our desired code parameters and has a minimal number of nonzero coefficients. The output is 64 bits which makes consecutive outputs easy to maintain in a packed array of 64-bit words. We note that if the PCLMUL instruction is available, we can apply the generator polynomial on 127 rounded output bits immediately by xoring outputs of such two instructions.

7.6 Conclusions and Future Directions

In summary, our main contributions in this chapter are:

1. Two candidate pseudorandom function families called SPRING-BCH and SPRING-CRT, which is short for "subset-product with rounding over a ring."

2. A concrete security analysis of the "learning with rounding" (LWR) problems that underlie our constructions. In particular, we formulate and analyze a new

"hidden" variant of learning with rounding/errors, in which a previously public component of the input is now partly hidden. This small change appears to significantly increase the concrete hardness of the problem, and in particular it disables all of the known non-trivial algorithms that outperform exponential brute-force search, namely generalized birthday attacks [95, 15] and lattice reduction attacks [71]. We believe that hidden LWR is an excellent target for further research.

We stress that in this chapter, we focus only on basic (fixed output length) PRFs, and in particular we do not specifically consider objects like unbounded stream ciphers, block ciphers, MACs, etc. (Note, however, that it is possible to construct such objects relatively efficiently from fixed-length PRFs, e.g., the Feistel/Luby-Rackoff transformation and variants [59, 75] from a PRF to a block cipher.) Also, at this point we only consider basic distinguishing attacks, and not more advanced ones such as side-channel, related-key, or forward-security attacks. These are all very interesting topics for further research.

Following up on this, Brenner *et al.* [27] implement SPRING-BCH directly on FPGA, and explore the vulnerability of the design to side-channel attacks. They manage to achieve speeds that outperform the software implementation from this chapter by a factor of 10, whilst using only 4% of the available resources on a typical FPGA board.

7.7 Cyclic Decomposition of R_2^*

In this section, we provide the necessary mathematical background needed to prove the decomposition theorem for R_2, Theorem 7.3.1. As such, this is not used elsewhere in the chapter, and readers may choose to skip it, if they wish.

We recall the polynomial ring $R = \mathbb{Z}[x]/\langle 1 + x^n \rangle$ where n is a power of 2, and the quotient ring $R_q = R/qR$ for any integer q. We denote the multiplicative group of

units in any ring T by T^*.

The fundamental theorem of finite abelian groups says that any such group can be written uniquely (up to order) as the direct product of cyclic groups of prime-power order. We let C_k denote the (multiplicative) cyclic group of order k, and C_k^ℓ denote the ℓ-fold direct product of C_k.

The ideal $\mathfrak{p} = \langle 2, 1+x\rangle$ in R is prime and has norm 2, i.e., $|R/\mathfrak{p}| = 2$, in fact $R/\mathfrak{p} = \{\mathfrak{p}, 1+\mathfrak{p}\}$. It follows that $\mathfrak{p}^k = \langle 2, (1+x)^k\rangle$ and $|R/\mathfrak{p}^k| = 2^k$. An element $a \in R$ is invertible modulo $\mathfrak{p}^k$ if and only if $a(1) = 1 \bmod \mathfrak{p}$, which implies that exactly half the elements of $R/\mathfrak{p}^k$ are invertible, and so the multiplicative group $(R/\mathfrak{p}^k)^*$ of units modulo $\mathfrak{p}^k$ has order $|(R/\mathfrak{p}^k)^*| = 2^{k-1}$. Also, since $(1+x)^{2^j} = 1 + x^{2^j} \bmod 2$ (because $\binom{2^j}{i}$ is even for every $0 < i < 2^j$), we have that $\mathfrak{p}^{2^j} = \langle 2, 1+x^{2^j}\rangle$ for $1 \le 2^j \le n$, and in particular, $\mathfrak{p}^n = \langle 2, 1+x^n\rangle = \langle 2\rangle$. We summarize this discussion as follows.

Claim 7.7.1. $R_2 = R/\mathfrak{p}^n$. $|R_2| = 2^n$. $|R_2^*| = 2^{n-1}$. In R_2 for any $k = 2^j$ (and in particular, for $k = n$) the ideal $\mathfrak{p}^k = \langle 2, (1+x)^k\rangle = \langle 2, 1+x^k\rangle$.

For any $a \in R$ that is invertible modulo an ideal I, define $\mathrm{ord}_{\mathcal{I}}(a)$ to be the order of a modulo $\mathcal{I}$, i.e., the least positive integer e such that $a^e = 1 \bmod \mathcal{I}$. We typically take $\mathcal{I} = \mathfrak{p}^k$ for some $k \ge 1$.

Our goal in this section is to determine the cyclic decomposition, and an explicit set of corresponding generators, of the multiplicative unit group R_2^*. We start with a simple fact relating the orders of elements modulo $\mathfrak{p}^k$ for different values of k.

Lemma 7.7.2. For $j \ge 0$, $1 \le k \le n$, and any $a \in R$, we have $a^j = 1 \bmod \mathfrak{p}^k$ if and only if $a^{2j} = 1 \bmod \mathfrak{p}^{2k}$. In particular, if $\mathrm{ord}_{\mathfrak{p}^{2k}}(a) > 1$ (i.e., if $a \neq 1 \bmod \mathfrak{p}^{2k}$) then $\mathrm{ord}_{\mathfrak{p}^{2k}}(a) = 2\,\mathrm{ord}_{\mathfrak{p}^k}(a)$.

Proof. First observe that

$$(a^j - 1)^2 = (a^{2j} - 1) - 2(a^j - 1). \tag{7.7.2}$$

If $a^j - 1 \in \mathfrak{p}^k$, then $(a^j-1)^2 \in \mathfrak{p}^{2k}$ and $2(a^j-1) \in \mathfrak{p}^{n+k} \subseteq \mathfrak{p}^{2k}$, so $a^{2j}-1 \in \mathfrak{p}^{2k}$ also.

For the other direction, suppose for the purpose of contradiction that $a^{2j}-1 \in \mathfrak{p}^{2k}$ but $a^j - 1 \in \mathfrak{p}^i \backslash \mathfrak{p}^{i+1}$ for some nonnegative $i < k \leq n$. Then because both $2k$ and $n+i$ are at least $2i+1$, the right-hand side of Equation (7.7.2) is in $\mathfrak{p}^{2i+1}$, whereas the left-hand side is not, a contradiction.

For the second claim, it suffices to observe that if $\mathrm{ord}_{\mathfrak{p}^{2k}}(a)$ is greater than 1, then it must be even, because it must divide $|(R/\mathfrak{p}^{2k})^*|$, which is a power of 2. □

Lemma 7.7.3. For any $j \geq 0$, $k > n$, and $a \in R$, we have $a^j = \pm 1 \bmod \mathfrak{p}^k$ if and only if $a^{2j} = 1 \bmod \mathfrak{p}^{n+k}$. In particular, if $\mathrm{ord}_{\mathfrak{p}^k}(a) > 1$ (i.e., if $a \neq 1 \bmod \mathfrak{p}^k$) then $\mathrm{ord}_{\mathfrak{p}^{n+k}}(a) = \mathrm{ord}_{\mathfrak{p}^k}(a)$ when $a^{\mathrm{ord}_{\mathfrak{p}^k}(a)/2} = -1 \bmod \mathfrak{p}^k$, and $\mathrm{ord}_{\mathfrak{p}^{n+k}}(a) = 2\,\mathrm{ord}_{\mathfrak{p}^k}(a)$ otherwise.

Proof. We have

$$(a^j \pm 1)^2 = (a^{2j} - 1) \pm 2(a^j \pm 1).$$

If $a^j \pm 1 \in \mathfrak{p}^k$, then $(a^j \pm 1)^2 \in \mathfrak{p}^{2k} \subseteq \mathfrak{p}^{n+k} = \mathfrak{p}^{n+k}$ and $2(a^j \pm 1) \in 2\mathfrak{p}^k = \mathfrak{p}^{n+k}$, so $a^{2j} - 1 \in \mathfrak{p}^{n+k}$ also.

For the other direction, suppose that

$$a^{2j} - 1 = (a^j + 1)(a^j - 1) \in \mathfrak{p}^{n+k}. \tag{7.7.4}$$

Now because $(a^j+1)-(a^j-1) = 2$, the largest power of $\mathfrak{p}$ that can divide both a^j+1 and a^j-1 is $\mathfrak{p}^n = \langle 2 \rangle$. Because $\mathfrak{p}$ is prime, it follows from Equation (7.7.4) that $\mathfrak{p}^k$ must divide one of a^j+1 or a^j-1, as desired.

To prove the second part of the lemma, we know that $\mathrm{ord}_{\mathfrak{p}^{n+k}}(a) \geq \mathrm{ord}_{\mathfrak{p}^k}(a)$ (because $a^e = 1 \bmod \mathfrak{p}^{n+k}$ implies $a^e = 1 \bmod \mathfrak{p}^k$), and also that $\mathrm{ord}_{\mathfrak{p}^{n+k}}(a) \leq 2\,\mathrm{ord}_{\mathfrak{p}^k}(a)$ by the first part of the lemma. Since both $\mathrm{ord}_{\mathfrak{p}^{k+n}}(a)$ and $\mathrm{ord}_{\mathfrak{p}^k}(a)$ are powers of two, we therefore have $\mathrm{ord}_{\mathfrak{p}^{n+k}}(a) \in \{1,2\} \cdot \mathrm{ord}_{\mathfrak{p}^k}(a)$. The claim then follows immediately from the first part of the lemma. □

We now proceed to establish the group structure of R_2^*, and also provide a set of independent generators. Lemma 7.7.4 gives the cyclic decomposition of $(R/\mathfrak{p}^{2^j})^*$ for $2 \leq 2^j \leq 2n$; because $\mathfrak{p}^n = \langle 2 \rangle$ this in particular gives the decompositions of R_2^*. Lemma 7.7.5 gives an independent set of generators for this group.

Lemma 7.7.4. For $2 \leq 2^j \leq 2n$, $(R/\mathfrak{p}^{2^j})^*$ decomposes into 2^{j-1} cyclic components as follows:

$$(R/\mathfrak{p}^{2^j})^* \cong C_2^{2^{j-2}} \times C_4^{2^{j-3}} \times \cdots \times C_{2^{j-1}}^1 \times C_{2^j}^1 = \prod_{i=1}^{j-1} C_{2^i}^{2^{j-i-1}} \times C_{2^j}.$$

Proof. We proceed by induction on j. For the base case, it is easy to see that for $2^j = 2$, we have $(R/\mathfrak{p}^2)^* = \{1, x\} \cong C_2$.

Now assume the inductive hypothesis for $2^j \leq n$, and let h be the natural homomorphism from $G = (R/\mathfrak{p}^{2^{j+1}})^*$ to $G' = (R/\mathfrak{p}^{2^j})^*$, defined as $h(a) = a \bmod \mathfrak{p}^{2^j}$. By the fundamental theorem, we can write $G \cong \prod_i S_{k_i}$, where each S_{k_i} is a nontrivial cyclic subgroup of G whose order k_i is a power of 2. By Lemma 7.7.2, $h(S_{k_i})$ is a (possibly trivial) cyclic subgroup of G', of order $k_i/2$. Moreover, because h is surjective and by definition of direct product, we have $G' = h(G) = \prod_i h(S_{k_i}) \cong \prod_i C_{k_i/2}$.

Now by the inductive hypothesis, we know that $G' \cong \prod_{i=1}^{j-1} C_{2^i}^{2^{j-i-1}} \times C_{2^j}$. Then by uniqueness of the cyclic decomposition, and recalling that the C_2 components of G become trivial under h, we therefore have

$$G \cong C_2^\ell \times \prod_{i=1}^{j-1} C_{2^{i+1}}^{2^{j-i-1}} \times C_{2^{j+1}} = C_2^k \times \prod_{i=2}^{(j+1)-1} C_{2^i}^{2^{(j+1)-i-1}} \times C_{2^{j+1}}$$

for some $\ell \geq 0$. Since G' has 2^{j-1} cyclic components, whose orders are half those of the corresponding components in G, we see that ℓ must satisfy

$$2^\ell = |G|/(|G'| \cdot 2^{2^{j-1}}) = 2^{2^{j+1}-1}/(2^{2^j-1} \cdot 2^{2^{j-1}}) = 2^{2^{j-1}},$$

which completes the proof. □

To obtain explicit generators of the unit groups, we define the following elements of R:

$$g_{0,0} = 1 + (1+x)^1;$$
$$g_{i,k} = 1 + (1+x)^{2^i+k} \quad i \geq 1, \textit{odd } k \in \{1, \ldots, 2^i\}.$$

Note that these elements are sparse (i.e., they have few monomials), which will be important later on for efficiently computing multi-products. We also notice that $\{g_{i,k}\}_{0 \leq i \leq \lg(n)} = \{1 + (1+x)^k\}_{\textit{odd } k \in \{1,\ldots,2n\}}$.

Lemma 7.7.5. For $2^j \leq 2n$, the set $\{g_{i,k}\}_{0 \leq i < j}$ is an independent set of generators of the group $(R/\mathfrak{p}^{2^j})^*$. Each $g_{i,k}$ has order 2^{j-i}, i.e., it generates a $C_{2^{j-i}}$ component of group's cyclic decomposition.

Proof. We proceed by induction on j, somewhat similarly to the proof of Lemma 7.7.4. For the base case $2^j = 2$, we have $(R/\mathfrak{p}^2)^* = \{1, x\}$, which is generated by $x = g_{0,0} \pmod 2$, which has order 2.

We now assume the inductive hypothesis for $2^j < n$, and prove the claim for $2^{j+1} \leq n$. We first show that for $G = (R/\mathfrak{p}^{2^{j+1}})^*$, the elements $g_{i,k}$ for $0 \leq i \leq j$ have the claimed orders $2^{(j+1)-i}$. For $0 \leq i < j$, by the inductive hypothesis and Lemma 7.7.2, we have

$$\mathrm{ord}_{\mathfrak{p}^{2^j}}(g_{i,k}) = 2^{j-i} \Rightarrow \mathrm{ord}_{\mathfrak{p}^{2^{j+1}}}(g_{i,k}) = 2^{(j+1)-i}.$$

For $i = j$, since $g_{j,k} = 1 \bmod \mathfrak{p}^{2^j}$ (because $\mathfrak{p}^{2^j} = \langle 2, 1+x^{2^j} \rangle$), we have $\mathrm{ord}_{\mathfrak{p}^{2^{j+1}}}(g_{j,k}) = 2$.

Since the number of elements $g_{i,k}$ (for $0 \leq i \leq j$) and their orders match the cyclic decomposition of G from Lemma 7.7.4, to complete the proof it suffices to show that these elements generate all of G. Towards this end, recall the natural homomorphism $h(a) = a \bmod \mathfrak{p}^{2^j}$ from G to $G' = (R/\mathfrak{p}^{2^j})^*$. Observe that its kernel $S = \ker(h)$ is the subgroup of square roots of unity in G, because $a \in \ker(h)$ if and only if

$a^2 = 1 \bmod \mathfrak{p}^{2^{j+1}}$, by Lemma 7.7.2. By the inductive hypothesis, we can express $h(a)$ as a product of (powers of) the elements $g_{i,k}$ for $0 \leq i < j$, modulo $\mathfrak{p}^{2^j}$. Define $\overline{a} \in G$ to be the same product, but modulo $\mathfrak{p}^{2^{j+1}}$, so that $h(a) = h(\overline{a})$ and thus $a/\overline{a} \in S$.

It therefore remains to prove that the $g_{i,k}$ for $0 \leq i \leq j$ generate the subgroup S; in fact, we will show that the powers $s_{i,k} = g_{i,k}^{2^{j-i}} = 1 + (1+x)^{2^j + k \cdot 2^{j-i}} \pmod{\mathfrak{p}^{2^{j+1}}}$ do so. To see this, we first observe that there is a group isomorphism from $(R/\mathfrak{p}^{2^j}, +)$ to $(S, \cdot)$ given by $f(a') = 1 + (1+x)^{2^j} \cdot a'$. Indeed, every $f(a') \in S$ by Lemma 7.7.2, and f is bijective and a homomorphism by inspection. We next observe that $f((1+x)^{k'}) = s_{i,k}$ for $0 \leq k' < 2^j$, where $k' = k \cdot 2^{j-i}$ for odd $k < 2^i$ when $k' > 0$ and $i = k = 0$ when $k' = 0$.

It finally remains to prove that $(1+x)^{k'}$ for $0 \leq k' < 2^j$ additively generate $(R/\mathfrak{p}^{2^j})$. Firstly, notice that $(R/\mathfrak{p}^{2^j})$ can be additively generated by $x^{k'}$ for $0 \leq k' < 2^j$. Next, we show by induction that any $x^{k'}$ for $0 \leq k' < 2^j$ can be additively generated by $(1+x)^k$ for $0 \leq k \leq k'$, and this will suffice to complete the proof. For $k' = 0$, $x^{k'} = (1+x)^{k'}$. For $k' > 0$, observe that the coefficient of $x^{k'}$ in the binomial expansion of $(1+x)^{k'} \bmod \mathfrak{p}^{2^j}$ is always 1. For every $k < k'$ such that the coefficient of x^k in the expansion is nonzero, we can always 'erase' this term by adding in a corresponding x^k term, which can be written as a sum of $(1+x)^\ell$ terms for $\ell \leq k < k'$ by hypothesis. Doing this for all $k < k'$ gives us an expression for $x^{k'}$ in terms of $(1+x)^k$ for $k \leq k'$, and this completes the induction, and the proof. $\square$

REFERENCES

[1] Agrawal, S., Boneh, D., and Boyen, X., "Efficient lattice (H)IBE in the standard model," in *EUROCRYPT*, pp. 553–572, 2010.

[2] Agrawal, S., Boneh, D., and Boyen, X., "Lattice basis delegation in fixed dimension and shorter-ciphertext hierarchical IBE," in *CRYPTO*, pp. 98–115, 2010.

[3] Ajtai, M., "Generating hard instances of lattice problems," *Quaderni di Matematica*, vol. 13, pp. 1–32, 2004. Preliminary version in STOC 1996.

[4] Alberini, G. and Rosen, A., "Efficient rounding procedures of biased samples." Manuscript, 2013.

[5] Alwen, J., Krenn, S., Pietrzak, K., and Wichs, D., "Learning with rounding, revisited - new reduction, properties and applications," in *CRYPTO*, pp. 57–74, 2013.

[6] Applebaum, B., Cash, D., Peikert, C., and Sahai, A., "Fast cryptographic primitives and circular-secure encryption based on hard learning problems," in *CRYPTO*, pp. 595–618, 2009.

[7] Arora, S. and Ge, R., "New algorithms for learning in presence of errors," in *Automata, Languages and Programming - 38th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4-8, 2011, Proceedings, Part I*, pp. 403–415, 2011.

[8] Banaszczyk, W., "Inequalites for convex bodies and polar reciprocal lattices in R^n," *Discrete & Computational Geometry*, vol. 13, pp. 217–231, 1995.

[9] Banerjee, A., Brenner, H., Leurent, G., Peikert, C., and Rosen, A., "SPRING: fast pseudorandom functions from rounded ring products," in *Fast Software Encryption - 21st International Workshop, FSE 2014, London, UK, March 3-5, 2014. Revised Selected Papers*, pp. 38–57, 2014.

[10] Banerjee, A., Fuchsbauer, G., Peikert, C., Pietrzak, K., and Stevens, S., "Key-homomorphic constrained pseudorandom functions," in *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II*, pp. 31–60, 2015.

[11] Banerjee, A. and Peikert, C., "New and improved key-homomorphic pseudorandom functions," in *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, pp. 353–370, 2014.

[12] Banerjee, A., Peikert, C., and Rosen, A., "Pseudorandom functions and lattices," in *EUROCRYPT*, pp. 719–737, 2012.

[13] Bellare, M. and Cash, D., "Pseudorandom functions and permutations provably secure against related-key attacks," in *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, pp. 666–684, 2010.

[14] Bellare, M. and Goldwasser, S., "New paradigms for digital signatures and message authentication based on non-interative zero knowledge proofs," in *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, pp. 194–211, 1989.

[15] Blum, A., Kalai, A., and Wasserman, H., "Noise-tolerant learning, the parity problem, and the statistical query model," *J. ACM*, vol. 50, no. 4, pp. 506–519, 2003.

[16] Blum, M. and Micali, S., "How to generate cryptographically strong sequences of pseudo-random bits," *SIAM J. Comput.*, vol. 13, no. 4, pp. 850–864, 1984.

[17] Boneh, D., Gentry, C., Gorbunov, S., Halevi, S., Nikolaenko, V., Segev, G., Vaikuntanathan, V., and Vinayagamurthy, D., "Fully key-homomorphic encryption, arithmetic circuit abe and compact garbled circuits," in *EUROCRYPT*, 2014.

[18] Boneh, D., Lewi, K., Montgomery, H. W., and Raghunathan, A., "Key homomorphic PRFs and their applications," in *CRYPTO*, pp. 410–428, 2013.

[19] Boneh, D., Montgomery, H. W., and Raghunathan, A., "Algebraic pseudorandom functions with improved efficiency from the augmented cascade," in *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA, October 4-8, 2010*, pp. 131–140, 2010.

[20] Boneh, D. and Waters, B., "Constrained pseudorandom functions and their applications," in *Advances in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part II*, pp. 280–300, 2013.

[21] Boyle, E., Goldwasser, S., and Ivan, I., "Functional signatures and pseudorandom functions," in *Public-Key Cryptography - PKC 2014 - 17th International Conference on Practice and Theory in Public-Key Cryptography, Buenos Aires, Argentina, March 26-28, 2014. Proceedings*, pp. 501–519, 2014.

[22] Brakerski, Z., Gentry, C., and Vaikuntanathan, V., "(Leveled) fully homomorphic encryption without bootstrapping," in *ITCS*, pp. 309–325, 2012.

[23] Brakerski, Z., Langlois, A., Peikert, C., Regev, O., and Stehlé, D., "Classical hardness of learning with errors," in *STOC*, pp. 575–584, 2013.

[24] Brakerski, Z. and Vaikuntanathan, V., "Efficient fully homomorphic encryption from (standard) LWE," in *FOCS*, pp. 97–106, 2011.

[25] Brakerski, Z. and Vaikuntanathan, V., "Fully homomorphic encryption from ring-LWE and security for key dependent messages," in *CRYPTO*, pp. 505–524, 2011.

[26] Brakerski, Z. and Vaikuntanathan, V., "Lattice-based FHE as secure as PKE," in *ITCS*, pp. 1–??, 2014.

[27] Brenner, H., Gaspar, L., Leurent, G., Rosen, A., and Standaert, F., "FPGA implementations of SPRING - and their countermeasures against side-channel attacks," in *Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings*, pp. 414–432, 2014.

[28] Cash, D., Hofheinz, D., Kiltz, E., and Peikert, C., "Bonsai trees, or how to delegate a lattice basis," in *EUROCRYPT*, pp. 523–552, 2010.

[29] Chen, Y. and Nguyen, P. Q., "BKZ 2.0: Better lattice security estimates," in *ASIACRYPT*, pp. 1–20, 2011.

[30] Dodis, Y. and Yampolskiy, A., "A verifiable random function with short proofs and keys," in *Public Key Cryptography - PKC 2005, 8th International Workshop on Theory and Practice in Public Key Cryptography, Les Diablerets, Switzerland, January 23-26, 2005, Proceedings*, pp. 416–431, 2005.

[31] "eBACS: ECRYPT Benchmarking of Cryptographic Systems." `http://bench.cr.yp.to`, accessed 11 November 2013.

[32] Freedman, M. J., Ishai, Y., Pinkas, B., and Reingold, O., "Keyword search and oblivious pseudorandom functions," in *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*, pp. 303–324, 2005.

[33] Garg, S., Gentry, C., Halevi, S., Raykova, M., Sahai, A., and Waters, B., "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*, pp. 40–49, 2013.

[34] Gentry, C., "Fully homomorphic encryption using ideal lattices," in *STOC*, pp. 169–178, 2009.

[35] Gentry, C., Peikert, C., and Vaikuntanathan, V., "Trapdoors for hard lattices and new cryptographic constructions," in *STOC*, pp. 197–206, 2008.

[36] Gentry, C., Sahai, A., and Waters, B., "Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based," in *CRYPTO*, pp. 75–92, 2013.

[37] Goldreich, O., "Two remarks concerning the goldwasser-micali-rivest signature scheme," in *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, pp. 104–110, 1986.

[38] Goldreich, O., Goldwasser, S., and Micali, S., "On the cryptographic applications of random functions," in *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*, pp. 276–288, 1984.

[39] Goldreich, O., Goldwasser, S., and Micali, S., "How to construct random functions," *J. ACM*, vol. 33, no. 4, pp. 792–807, 1986.

[40] Goldwasser, S., Kalai, Y. T., Peikert, C., and Vaikuntanathan, V., "Robustness of the learning with errors assumption," in *ICS*, pp. 230–240, 2010.

[41] Gorbunov, S., Vaikuntanathan, V., and Wee, H., "Attribute-based encryption for circuits," in *STOC*, pp. 545–554, 2013.

[42] Håstad, J., Impagliazzo, R., Levin, L. A., and Luby, M., "A pseudorandom generator from any one-way function," *SIAM J. Comput.*, vol. 28, no. 4, pp. 1364–1396, 1999.

[43] Hohenberger, S. and Waters, B., "Constructing verifiable random functions with large input spaces," 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*, pp. 656–672, 2010.

[44] Hopper, N. J. and Blum, M., "Secure human identification protocols," in *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, pp. 52–66, 2001.

[45] Jarecki, S. and Liu, X., "Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection," in *Theory of Cryptography, 6th Theory of Cryptography Conference, TCC 2009, San Francisco, CA, USA, March 15-17, 2009. Proceedings*, pp. 577–594, 2009.

[46] Juels, A. and Weis, S. A., "Authenticating pervasive devices with human protocols," in *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, pp. 293–308, 2005.

[47] Käsper, E. and Schwabe, P., "Faster and Timing-Attack Resistant AES-GCM," in *CHES*, pp. 1–17, 2009.

[48] Katz, J., Shin, J. S., and Smith, A., "Parallel and concurrent security of the HB and hb^+ protocols," *J. Cryptology*, vol. 23, no. 3, pp. 402–421, 2010.

[49] Kiayias, A., Papadopoulos, S., Triandopoulos, N., and Zacharias, T., "Delegatable pseudorandom functions and applications," in *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pp. 669–684, 2013.

[50] Kiltz, E., Pietrzak, K., Cash, D., Jain, A., and Venturi, D., "Efficient authentication from hard learning problems," in *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, pp. 7–26, 2011.

[51] Lenstra, A. K., Lenstra, Jr., H. W., and Lovász, L., "Factoring polynomials with rational coefficients," *Mathematische Annalen*, vol. 261, pp. 515–534, December 1982.

[52] Leurent, G., Bouillaguet, C., and Fouque, P.-A., "SIMD Is a Message Digest." Submission to NIST, 2008. `http://www.di.ens.fr/ leurent/files/SIMD.pdf`.

[53] Lewi, K., Montgomery, H. W., and Raghunathan, A., "Improved constructions of prfs secure against related-key attacks," in *ACNS*, pp. 44–61, 2014.

[54] Lewko, A. B. and Waters, B., "Efficient pseudorandom functions from the decisional linear assumption and weaker variants," in *Proceedings of the 2009 ACM Conference on Computer and Communications Security, CCS 2009, Chicago, Illinois, USA, November 9-13, 2009*, pp. 112–120, 2009.

[55] Lindner, R. and Peikert, C., "Better key sizes (and attacks) for LWE-based encryption," in *CT-RSA*, pp. 319–339, 2011.

[56] Liskov, M., "Updatable zero-knowledge databases," in *Advances in Cryptology - ASIACRYPT 2005, 11th International Conference on the Theory and Application of Cryptology and Information Security, Chennai, India, December 4-8, 2005, Proceedings*, pp. 174–198, 2005.

[57] Liu, M. and Nguyen, P. Q., "Solving BDD by enumeration: An update," in *CT-RSA*, pp. 293–309, 2013.

[58] Luby, M., *Pseudorandomness and cryptographic applications*. Princeton computer science notes, Princeton University Press, 1996.

[59] Luby, M. and Rackoff, C., "How to construct pseudorandom permutations from pseudorandom functions," *SIAM J. Comput.*, vol. 17, no. 2, pp. 373–386, 1988.

[60] Lyubashevsky, V. and Micciancio, D., "Generalized compact knapsacks are collision resistant," in *ICALP (2)*, pp. 144–155, 2006.

[61] Lyubashevsky, V., Micciancio, D., Peikert, C., and Rosen, A., "SWIFFT: A modest proposal for FFT hashing," in *FSE*, pp. 54–72, 2008.

[62] Lyubashevsky, V., Peikert, C., and Regev, O., "On ideal lattices and learning with errors over rings," *Journal of the ACM*, vol. 60, pp. 43:1–43:35, November 2013. Preliminary version in EUROCRYPT '10.

[63] Lyubashevsky, V., Peikert, C., and Regev, O., "A toolkit for ring-LWE cryptography," in *EUROCRYPT*, pp. 35–54, 2013.

[64] Mereghetti, C. and Palano, B., "Threshold circuits for iterated matrix product and powering," *ITA*, vol. 34, no. 1, pp. 39–46, 2000.

[65] Micali, S., Rabin, M. O., and Vadhan, S. P., "Verifiable random functions," in *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pp. 120–130, 1999.

[66] Micali, S. and Reyzin, L., "Soundness in the public-key model," in *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, pp. 542–565, 2001.

[67] Micali, S. and Rivest, R. L., "Micropayments revisited," in *Topics in Cryptology - CT-RSA 2002, The Cryptographer's Track at the RSA Conference, 2002, San Jose, CA, USA, February 18-22, 2002, Proceedings*, pp. 149–163, 2002.

[68] Micciancio, D., "Generalized compact knapsacks, cyclic lattices, and efficient one-way functions," *Computational Complexity*, vol. 16, no. 4, pp. 365–411, 2007. Preliminary version in FOCS 2002.

[69] Micciancio, D. and Mol, P., "Pseudorandom knapsacks and the sample complexity of LWE search-to-decision reductions," in *CRYPTO*, pp. 465–484, 2011.

[70] Micciancio, D. and Peikert, C., "Trapdoors for lattices: Simpler, tighter, faster, smaller," in *EUROCRYPT*, pp. 700–718, 2012.

[71] Micciancio, D. and Regev, O., "Lattice-based cryptography," in *Post Quantum Cryptography*, pp. 147–191, Springer, February 2009.

[72] Micciancio, D. and Voulgaris, P., "Faster exponential time algorithms for the shortest vector problem," in *SODA*, pp. 1468–1480, 2010.

[73] Naor, J. and Naor, M., "Small-bias probability spaces: Efficient constructions and applications," *SIAM J. Comput.*, vol. 22, no. 4, pp. 838–856, 1993.

[74] Naor, M., Pinkas, B., and Reingold, O., "Distributed pseudo-random functions and kdcs," in *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, pp. 327–346, 1999.

[75] Naor, M. and Reingold, O., "On the construction of pseudorandom permutations: Luby-rackoff revisited," *J. Cryptology*, vol. 12, no. 1, pp. 29–66, 1999.

[76] Naor, M. and Reingold, O., "Synthesizers and their application to the parallel construction of pseudo-random functions," *J. Comput. Syst. Sci.*, vol. 58, no. 2, pp. 336–375, 1999.

[77] Naor, M. and Reingold, O., "Number-theoretic constructions of efficient pseudo-random functions," *J. ACM*, vol. 51, no. 2, pp. 231–262, 2004.

[78] Naor, M., Reingold, O., and Rosen, A., "Pseudorandom functions and factoring," *SIAM J. Comput.*, vol. 31, no. 5, pp. 1383–1404, 2002.

[79] Pass, R., Seth, K., and Telang, S., "Indistinguishability obfuscation from semantically-secure multilinear encodings," in *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, pp. 500–517, 2014.

[80] Peikert, C., "Public-key cryptosystems from the worst-case shortest vector problem," in *STOC*, pp. 333–342, 2009.

[81] Peikert, C., "An efficient and parallel Gaussian sampler for lattices," in *CRYPTO*, pp. 80–97, 2010.

[82] Peikert, C. and Rosen, A., "Efficient collision-resistant hashing from worst-case assumptions on cyclic lattices," in *TCC*, pp. 145–166, 2006.

[83] Peikert, C. and Waters, B., "Lossy trapdoor functions and their applications," in *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pp. 187–196, 2008.

[84] Pietrzak, K., "Subspace LWE," in *TCC*, pp. 548–563, 2012.

[85] Razborov, A. A. and Rudich, S., "Natural proofs," *J. Comput. Syst. Sci.*, vol. 55, no. 1, pp. 24–35, 1997.

[86] Regev, O., "On lattices, learning with errors, random linear codes, and cryptography," *J. ACM*, vol. 56, no. 6, pp. 1–40, 2009. Preliminary version in STOC 2005.

[87] Reif, J. H. and Tate, S. R., "On threshold circuits and polynomial computation," *SIAM J. Comput.*, vol. 21, no. 5, pp. 896–908, 1992.

[88] Sahai, A. and Waters, B., "How to use indistinguishability obfuscation: deniable encryption, and more," in *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pp. 475–484, 2014.

[89] Schnorr, C.-P., "A hierarchy of polynomial time lattice basis reduction algorithms," *Theor. Comput. Sci.*, vol. 53, pp. 201–224, 1987.

[90] Shor, P. W., "Algorithms for quantum computation: Discrete logarithms and factoring," in *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, pp. 124–134, 1994.

[91] Valiant, L. G., "A theory of the learnable," *Commun. ACM*, vol. 27, no. 11, pp. 1134–1142, 1984.

[92] van de Pol, J. and Smart, N., "Estimating key sizes for high dimensional lattice-based systems," in *Cryptography and Coding* (Stam, M., ed.), vol. 8308 of *Lecture Notes in Computer Science*, pp. 290–303, Springer Berlin Heidelberg, 2013.

[93] van Dijk, M., Gentry, C., Halevi, S., and Vaikuntanathan, V., "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*, pp. 24–43, 2010.

[94] Vershynin, R., *Compressed Sensing, Theory and Applications*, ch. 5, pp. 210–268. Cambridge University Press, 2012. Available at `http://www-personal.umich.edu/ romanv/papers/non-asymptotic-rmt-plain.pdf`.

[95] Wagner, D., "A generalized birthday problem," in *CRYPTO*, pp. 288–303, 2002.