Logarithmic-Depth Pseudorandom Functions from Well-Founded Code-Based Assumptions

Youlong Ding* Aayush Jain† Ilan Komargodski‡

Abstract

We give the first NC^1-computable Pseudorandom Function (PRF) constructions from well-founded (non-ad-hoc) code-based assumptions. Specifically, we give two constructions based on two different classical variants of the learning parity with noise (LPN) assumption:

- An NC^1-computable PRF from hardness of Sparse-LPN [Alekhnovich, FOCS '03] (with respect to a sublinear-depth expander graph family). This PRF is also key-homomorphic.
- An NC^1-computable PRF from hardness of Ring-LPN [Heyse et al., FSE '12].

Both of these assumptions have been studied for many years in cryptography and average-case complexity. Ring-LPN has been used in the context of silent preprocessing for MPC and has a stable cryptanalysis. The study of counterexamples for Sparse-LPN is an active area of research.

Notably, within the range of parameters that we need, none of these assumptions is known to imply collision-resistant hashing, and the first is not known to imply public-key encryption. Prior constructions of code-based PRFs (let alone key-homomorphic) are either super-logarithmic depth, or rely on newly introduced assumptions.

As a bonus, we give a similar result relying on the *classical* LPN assumption, albeit with quasi-polynomial hardness:

- A key-homomorphic NC^1-computable PRF from quasi-polynomial hardness of the classical LPN [Blum et al., CRYPTO '93].

Technically, all of our results are obtained via a refinement and substantial extension of the recent ideas of [Ding, Jain, and Komargodski, STOC '25].

*The Hebrew University of Jerusalem, Israel. Email: youlong.ding@mail.huji.ac.il.

†Carnegie Mellon University, USA. Email: aayushja@andrew.cmu.edu.

‡The Hebrew University of Jerusalem, Israel. Email: ilank@cs.huji.ac.il.

Contents

1 Introduction **1**
- 1.1 Our Results . . . 3

2 Technical Overview **7**

3 Preliminaries **16**
- 3.1 Pseudorandom Functions . . . 18
- 3.2 Additional Notation for Matrix Operations . . . 19

4 A Unified Framework: MVP **20**
- 4.1 A Sufficient Condition for NC^1 PRFs . . . 20
- 4.2 Function Definition . . . 22
- 4.3 Evaluation Algorithm . . . 24
- 4.4 Efficiency Analysis . . . 27
- 4.5 Security Analysis . . . 30
- 4.6 Key-Homomorphism Analysis . . . 31
- 4.7 Putting Everything Together . . . 33

5 Instantiation I: NC^1 PRFs from Polynomial Hardness of SparseLPN **33**
- 5.1 The SparseLPN Assumption . . . 33
- 5.2 Walking on Expander Graphs . . . 35
- 5.3 Configuration . . . 37

6 Instantiation II: NC^1 PRFs from Polynomial Hardness of RingLPN **39**
- 6.1 The RingLPN Assumption . . . 39
- 6.2 Reduction to RingLPN with Sparse Secrets . . . 41
- 6.3 Bernoulli Sample Function over $\mathbb{F}_q$ . . . 42
- 6.4 Configuration . . . 42

7 Instantiation III: NC^1 PRFs from Quasi-Polynomial Hardness of LPN **45**
- 7.1 The LPN Assumption . . . 45
- 7.2 Reduction to LPN with Regular Sparse Coefficients . . . 45
- 7.3 Configuration . . . 46

A Omitted Proofs **49**
- A.1 Proof of Lemma 3 . . . 49
- A.2 Proof of Theorem 7 . . . 49
- A.3 Proof of Theorem 9 . . . 51
- A.4 Proof of Lemma 8 . . . 56
- A.5 Proof of Lemma 10 . . . 57
- A.6 Proof of Lemma 13 . . . 58

1 Introduction

Pseudorandom functions (PRFs) are a family of efficient keyed functions $f_{\mathbf{k}}\colon \{0,1\}^n \to \{0,1\}^{m(n)}$ such that for a randomly chosen key $\mathbf{k}$ (from a key space), $f_{\mathbf{k}}(\cdot)$ is indistinguishable from a random function $R(\cdot)$ by an efficient adversary who has oracle access to it. Since their introduction (Goldreich, Goldwasser, and Micali [GGM86]), PRFs have emerged as a cornerstone of modern cryptography, underpinning an extraordinary range of constructions across nearly every domain, as well as providing a tool to answer questions in complexity theory, learning, data structures, and more. Given this influence, understanding the inherent computational complexity of PRFs is a foundational challenge in cryptography.

In terms of theoretical feasibility, it is well known that PRFs are existentially equivalent to one-way functions [GGM86, HILL99], the minimal cryptographic assumption. However, this sequence of transformations incurs a non-trivial cost in efficiency [HRV13, VZ12, BMM24]. A central measure of efficiency of PRFs is their parallel complexity, or circuit depth. Often, the depth of PRFs directly affects efficiency of applications, beyond computational complexity. For instance, it affects round complexity of secure computation protocols [DI06, IKOS08, BJKL21], enables advanced cryptographic systems [GVW12, App14, Zha25], and gives applications in computational complexity [Val84, KV89, RR97, IL90, BFKL93] and meta-complexity [AN25]. Unfortunately, GGM's construction [GGM86] is not "depth-optimal:" it transforms a polynomial-stretch pseudorandom generator (PRG) on n input bits, with polynomial stretch, and computable in depth $d(n)$, into a PRF computable in depth $\omega(d(n) + \log n)$. It is worth noting that there is a generic transformation [IKOS08, FLY22], turning any PRF in NC^1 into an optimally efficient PRF with $(2+\epsilon)n$ size and $(1+\epsilon)\log n$ depth.

Constructions of lower-depth PRFs are known from almost any standard algebraic assumption. Naor, Reingold, and Rosen [NR97, NR04, NRR02] gave constructions in $\mathsf{TC}^0 \subseteq \mathsf{NC}^1$ from DDH and factoring. Lewko and Waters [LW09] gave a construction in TC^0 from the k-Linear assumption (which is possibly weaker than DDH). Banerjee, Peikert, and Rosen [BPR12] gave a construction in TC^0 from the ring variant of learning with errors assumption (Ring-LWE), where the modulus to noise ratio is super-polynomial. Viola [Vio15] gave a PRF in $\mathsf{AC}^0[\oplus]$ assuming Quasi-Polynomial Hardness of factoring.[1] See Table 1 for a list of currently known results. We refer to [BR17] for an elaborate overview of prior work in this area.

Towards Code-Based NC^1 PRFs. Code-based cryptography, dating back to [McE78], relies on the average-case hardness assumptions for decoding noisy linear codes. It is often referred to as Learning Parity with Noise (LPN) in the cryptography literature [BFKL93] and as syndrome decoding in the coding theory community [Pra62]. Given a field $\mathbb{F}$, let $\mathcal{C}$ be a code generation algorithm such that $\mathcal{C}(n,m)$ outputs a matrix in $\mathbb{F}^{n\times m}$. For $\mu \in (0,1)$, let $\mathsf{Ber}_\mu(\mathbb{F})$ denote the distribution which outputs a uniformly random nonzero element in $\mathbb{F}\backslash\{0\}$ with probability μ, and 0 with probability $1-\mu$. An LPN assumption with respect to $\mathcal{C}$ postulates that the following two

[1] TC^0 consists of all languages decidable by constant depth and polynomial size Boolean circuits, containing only unbounded fan-in AND gates, OR gates, NOT gates, and majority gates. NC^1 consists of all languages decidable by polynomial size and logarithmic depth circuits, containing constant fan-in AND, OR, and NOT gates. AC^0 consists of all languages decidable by polynomial size and constant depth circuits, containing unbounded fan-in AND, OR, and NOT gates. $\mathsf{AC}^0[m]$ is similar to AC^0 except that the circuits can additionally have unbounded fan-in MOD_m gates. ACC^0 is the union over all m of $\mathsf{AC}^0[m]$.

distributions are computationally indistinguishable:

$$(\mathbf{A}, \mathbf{s}\cdot\mathbf{A}+\mathbf{e}) \approx (\mathbf{A}, \mathbf{u}), \quad \text{where } \mathbf{A} \leftarrow \mathcal{C}(n,m) \in \mathbb{F}^{n\times m},\ \mathbf{s} \leftarrow\!\$\ \mathbb{F}^n,\ \mathbf{e} \leftarrow\!\$\ \mathsf{Ber}_\mu^m(\mathbb{F}),\ \mathbf{u} \leftarrow\!\$\ \mathbb{F}^m.$$

Informally, it postulates that noisy decoding of $\mathcal{C}$ is hard (viewing $\mathbf{s}$ as a message, and $\mathbf{s}\cdot\mathbf{A}+\mathbf{e}$ as a noisy codeword). A unique feature of LPN-style assumptions is that, as long as the underlying code $\mathcal{C}$ has a large dual distance, they are unconditionally secure against linear tests, a framework that captures many existing attacks on noisy linear-equation assumptions.

The standard LPN [BFKL93] is recovered by letting $\mathbb{F} = \mathbb{F}_2$ and $C(n,m)$ be the uniform distribution over $\mathbb{F}_2^{n\times m}$. Besides standard LPN, commonly studied variants (that are relevant to this work) include:

- RingLPN [HKL+12, BCG+20], where $\mathcal{C}$ outputs a matrix $\mathbf{A} = (\mathbf{A}_1, \ldots, \mathbf{A}_{m/n})$, such that each $\mathbf{A}_i \in \mathbb{F}_q^{n\times n}$ is induced by a random element in a degree-n cyclotomic polynomial ring $\mathcal{R}_q$. This assumption is closely related to the syndrome decoding of random quasi-negacyclic codes. It has received growing attention in recent years [BL12, DP12, LP15, GJL15, BCG+20, LWYY24] due to its appealing efficiency advantages over the classical version of LPN.

- SparseLPN [Ale03, Ale11], where $\mathcal{C}$ outputs a column-sparse matrix $\mathbf{A}$ such that $\mathbf{A}$ is (the adjacency matrix of) a good expander graph. The most common instantiation of SparseLPN uses a random expander graph, which corresponds to a random column-sparse matrix[2]. In this work, we will make use of sparse LPN assumption such that the neighbor function of the underlying expander graph can be computed in sublinear depth with respect to its input length (i.e., in depth $o(\log m)$ where m is the number of equations/number of left nodes in the (bipartite) expander graph). This is a very mild structural requirement because: i) the neighbour function itself can be allowed to have exponential circuit size, and, ii) a random expander graph has only slightly larger depth of $O(\log m)$.

  SparseLPN is closely related to the syndrome decoding of the duals of expander codes.[3] Since its introduction, it has been studied and used for over a couple of decades in cryptography and computational learning theory [AIK08, IKOS08, ABW10, ADI+17, JLS21, JLS22, DIJL23, JLS24, RVV24, CHKV25, BBTV25, DJK25, GHJS25, CSZ25]. It is also closely related to Goldreich's local function [Gol00, BQ12, OW14, App16, AR16], where instead of injecting fresh noise, a fixed, handcrafted predicate f is applied separately on each subset of $\mathbf{s}$ touched by each column of $\mathbf{A}$. We note that there are specific tailored constructions of expanders for which, under certain parameter regions, SparseLPN can be subject to efficient attacks [OST19, OST22, BLRS26]. Our parameter region does not fall into these known counterexamples, and our work should be viewed as providing a general framework that applies to any expander graph satisfying the sublinear-depth condition for which the corresponding SparseLPN assumption holds. In particular, understanding which specific expander families lead to counterexamples is an active and important research direction in cryptography and average-case complexity. This line of work is complementary to our goal: we identify structural conditions under which the SparseLPN assumption yields an NC^1 PRF.

[2] This relies on the fact that a random column-sparse matrix is a good expander with high probability.

[3] Recall that, in expander codes, the matrix $\mathbf{A}$ serves as the parity-check matrix. In our setting, $\mathbf{A}$ serves as the generator matrix, and hence the resulting code is the dual of an expander code. Therefore, a larger distance of the corresponding expander code translates into a larger dual distance for the code considered here, making it more secure against linear tests.

Key-Homomorphism. A PRF $f_{\mathbf{k}}$ is key-homomorphic (KH) if for *any two keys* $\mathbf{k_1}, \mathbf{k_2}$, and any input x, it holds that $f_{\mathbf{k_1} \oplus \mathbf{k_2}}(x) = f_{\mathbf{k_1}}(x) + f_{\mathbf{k_2}}(x)$ [NPR99, BLMR13]. KH-PRFs have many useful applications in symmetric cryptography and give rise to distributed PRFs, symmetric-key proxy re-encryption, and updatable encryption [BLMR13, EPRS17, LT18, KLR19, BDGJ20, BEKS20], OT extension [Sch18], and secure aggregation [LLPT23]. In the plain model (no random oracles), existing KH-PRFs constructions satisfy a relaxed notion called almost-key-homomorphism [BLMR13], where the homomorphism requirement is "approximately" correct, i.e., $f_{\mathbf{k_1} \oplus \mathbf{k_2}}(x) \in f_{\mathbf{k_1}}(x) + f_{\mathbf{k_2}}(x) \pm \delta$ for some (typically small) "error" parameter $\delta > 0$.[4]

A recent work of Ding, Jain, and Komargodski [DJK25] made significant progress towards logarithmic-depth (NC^1) PRFs from LPN-style assumptions. They give:

- A *weak* PRF computable in NC^1 from classical LPN. In a *weak* PRF the adversary is only given (input,output)-samples for a random key and uniformly random inputs.
- A (strong) PRF computable in $\mathsf{NC}^{1+\epsilon}$ from $\mathsf{SparseLPN}$ (with respect to a random expander).
- A (strong) PRF computable in NC^1 from the *super-polynomial hardness* of a *new, ad hoc* variant of the LPN assumption (called "seeded LPN")[5].

Despite this progress, it remains open whether one can obtain an NC^1-computable *strong* PRF from a *well-founded* code-based assumption. We ask:

Are there NC^1-computable PRFs based on well-founded code-based assumptions?

It is also worth noting that their constructions are key-homomorphic. To date, the only known constructions of key-homomorphic *strong* PRFs from *well-founded* assumptions are based on LWE (e.g., [BLMR13, BP14, Kim20]) and the aforementioned Sparse-LPN-based construction; however, none of these achieves NC^1. We further ask:

Are there NC^1-computable KH-PRFs based on well-founded assumptions?

1.1 Our Results

In this paper, we propose new constructions of PRFs that answer both of the above questions affirmatively. We refer to Table 1 for a summary. We show for the first time the possibility of building NC^1 and key-homomorphic PRFs from well-founded code-based assumptions. Our meta theorem is generic and can be instantiated from any LPN variants as long as the required conditions are met. All our constructions are instantiations of this framework.

Theorem 1 (Meta theorem for KH-PRFs, informal; see Theorem 6)**.** *Let λ be a security parameter. Let $n \leq \mathsf{poly}(\lambda)$, $m = n\lambda$. Let $\mu = \mu(n) \in (0, 1)$. Let $\mathbf{M} = (\mathbf{M}_1, \ldots, \mathbf{M}_\lambda) \in \mathbb{F}_q^{n \times m}$ where each matrix $\mathbf{M}_i \in \mathbb{F}_q^{n \times n}$. There exist KH-PRFs computable in NC^1 if there exists a distribution of matrices $\mathbf{M}$ such that:*

1. *Each column of $\mathbf{M}$ has a Hamming weight bounded by a parameter d such that $\log d = o(\log \lambda)$.*

[4] Approximate key-homomorphism suffices for all of the above mentioned applications.

[5] Specifically, they first build a PRG G with super-polynomial stretch and random access to its output. This is why a super-polynomial assumption is required there. The induced PRF is obtained by setting $F_k(i) := G(k)_i$.

2. *The noise rate is sufficiently low such that* $\mu \cdot d^{\omega(1)} = o(1)$.

3. *There exists a circuit of depth* $O(\log \lambda)$ *that computes iterated matrix multiplication for up to* $k = \omega(1)$ *matrices in* $\mathbf{M}$. *Namely, computing* $\prod_{i \in S} \mathbf{M}_i$ *where* $\{\mathbf{M}_i\}_{i \in S}$ *for* $|S| \leq k$ *is in depth* $O(\log \lambda)$.

4. *There exists a circuit of depth* $O(\log \lambda)$ *that computes the reachability set for up to* $k = \omega(1)$ *step walking in a graph induced by* $\mathbf{M}$. *Namely, for every block* $\mathbf{M}_i$ *we view it as a graph given by the adjacency* $\alpha \rightsquigarrow \beta$ *iff* $\mathbf{M}[\alpha, \beta] \neq 0$. *We require that there exists a circuit in depth* $O(\log \lambda)$ *that takes any node* $\gamma \in [n]$ *and computes the set of nodes obtained by walking on* k *graphs induced by matrices* $\{\mathbf{M}_i\}_{i \in S}$ *for* $|S| \leq k$.

5. *(LPN-style samples are pseudorandom) For any polynomial* $Q = \mathsf{poly}(\lambda)$, $(\mathbf{M}, \mathbf{S} \cdot \mathbf{M} + \mathbf{E}) \approx (\mathbf{M}, \mathbf{U})$. *where* $\mathbf{S} \leftarrow\!\$\ \mathbb{F}_q^{Q \times n}, \mathbf{M} \leftarrow\!\$\ \mathcal{M} \in \mathbb{F}_q^{n \times \lambda n}$, $\mathbf{E} \leftarrow\!\$\ \mathsf{Ber}_\mu^{Q \times \lambda \cdot n}, \mathbf{U} \leftarrow\!\$\ \mathbb{F}_q^{Q \times \lambda \cdot n}$.

In a nutshell, our Theorem 1 shows that if LPN with dimension n, polynomially large samples, and sufficiently small noise rate holds with respect to a distribution of sparse coefficient matrices for which the iterated product of some $\omega(1)$ matrices can be computed in $O(\log \lambda)$ depth, then there exists KH-PRFs in NC^1. For important technical reasons, our result needs an additional requirement about the graphs induced by such sparse matrices: Graph traversal can be done in low-depth. This condition is related but not equivalent to computability of iterated matrix product. This additional condition plays an important role for enabling a log-depth computation of PRFs. We will explain this more in the technical overview.

Similarly, we also consider a slightly weaker structural result, that does not yield key homomorphic PRFs, but does yield log-depth PRFs. In this meta-theorem, we relax the condition 5 by allowing $\mathbf{M}$ to be hidden.

Theorem 2 (Meta theorem for PRFs, informal; see Theorem 6)**.** *There exist PRFs computable in* NC^1 *assuming the same condition as in Theorem 1 are satisfied, except that we replace the condition 5 with the following:*

- *(the noisy low-rank matrix is pseudorandom)* $\mathbf{V} \cdot \mathbf{M} + \mathbf{E} \approx \mathbf{U}$ *(i.e., the matrix* $\mathbf{M}$ *is not public).*

We then obtain our results from the above meta theorems.

Result 1: PRF in NC^1 from $\mathsf{SparseLPN}$ Our first result is obtained by instantiating Theorem 1 with matrix $\mathbf{M}$ chosen to be the adjacency matrix of a sublinear-depth expander graph family.

Theorem 3 (Main result, informal; see Corollary 1)**.** *Let* λ *be a security parameter. There exist KH-PRFs computable by* NC^1 *circuits assuming hardness of* $\mathsf{SparseLPN}$ *(with respect to an expander graph family whose neighbor function is computable in sublinear depth) for dimension* $n = \lambda^{10}$, *number of samples* $m = n^{1.1}$, *noise rate* $\mu = n^{-1/\log\log n}$, *and sparsity* $d = \log^{10} n$.

Remark 1. We note that the above construction is the first NC^1 KH-PRF from *any* well-founded assumption. In addition, in the range of parameters we use, the underlying assumption is not known to imply collision-resistant hashing or public-key encryption.

Remark 2 (On our $\mathsf{SparseLPN}$ assumption)**.** The original $\mathsf{SparseLPN}$ assumption [Ale03] posits that the assumption holds with respect to any expander with sufficiently good expansion (assumption $(\star)$).

Assumption ($\star$) is unconditionally secure against linear tests. It was used in [GHJS25]; moreover, a closely related variant, in which the random noise is replaced by a deterministic predicate, was used in [AR16]. Recent works have provided cautionary evidence that ($\star$), in its full generality, might be false [OST19, OST22, BLRS26], by giving attacks for specific choices of expander families. The attacks of [OST19, OST22] apply when the expander is computable in $\mathsf{AC}^0[\oplus]$ and when a quasipolynomial number of samples is available. The attack of [BLRS26] is tailored to specific expander constructions: it exploits efficient decoding algorithms for Reed–Muller codes by relating the expanders of [GUV09] to Reed–Muller codes.

1. First, the existence of such tailored counterexamples should not be interpreted as weakening our result. This is analogous to other cryptographic assumptions, e.g., DDH over insecure groups, and Ring-LWE/Ring-LPN over insecure ring choices, but these facts do not invalidate constructions based on (well-chosen instantiations of) these assumptions. Similarly, our results require only the existence of a slightly low-depth expander family for which the corresponding SparseLPN assumption holds. Existing counterexamples are highly tailored to specific expander constructions, and we expect many other sublinear-depth expanders (e.g. [KT22]) to remain plausible candidates for SparseLPN.

2. Second, we note that our construction does not fall into the parameter regimes covered by these attacks. Our parameters are quite mild: we can work with any polylogarithmic sparsity, e.g. $\log^{10} n$, with $n^{1.1}$ samples and error rate as high as $n^{-1/\log\log n}$, which is much larger than the inverse polynomial noise rate required by [BLRS26]. Our regime is not currently ruled out by any known counterexample. The only structural condition we impose is that the expander's neighbor function be computable by a (possibly exponential-size) circuit of sublinear depth (measured with respect to its input length), i.e., depth $o(\log m)$ and size $\mathsf{poly}(m)$. This condition is only slightly more structured than a random expander, whose neighbor function can be implemented in depth $O(\log m)$.

3. Thus, the main point of our work is to identify the structural property (i.e., sublinear depth) of an expander family that is sufficient for obtaining PRFs in NC^1. Once SparseLPN holds for any such sublinear-depth expanders, our framework yields an NC^1 PRF. We hope that this perspective motivates further study of low-depth expanders.[6] In this sense, understanding which particular structured expander families admit attacks is complementary to our goal.

Result 2: PRF in NC^1 from RingLPN Our second result is obtained by instantiating Theorem 2 with matrix $\mathbf{M}$ chosen to have certain quasi-negacyclic structure (corresponds to a sparse polynomial over a cyclotomic ring $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$). Then we show that its security can be reduced to RingLPN, albeit at the cost of losing key homomorphism in this construction.

Theorem 4 (Informal; see Corollary 2)**.** *Let λ be a security parameter. There exist PRFs computable by NC^1 circuits assuming hardness of RingLPN for dimension $n = \lambda$, any polynomial number of samples, noise rate $\mu = \log^{10} n/n$.*

[6] Historically, the expander literature has primarily focused on minimizing size, whereas our application only requires low depth; the size can even be exponential.

Table 1: Known constructions of NC^1-computable PRFs. Newly conjectured/ad hoc/heuristic assumptions are highlighted with yellow background. **KH** stands for key homomorphism. t is the adversary running time, ϵ is the adversary advantage. For code-based assumptions, n is the dimension of the code, m is the length of the code, μ is the Bernoulli noise rate, d is the degree of the expander code. The constant polynomial power of 1.01 can be made arbitrarily close to 1.

Construction	Assumption	Category	KH
[NR97, NR04]	DDH	Group-Based	✗
[LW09]	k-Linear	Group-Based	✗
[NR97, NR04] [NRR02]	Factoring	Number Theory-Based	✗
[MV12, MV15]	Ad Hoc	SPN-Based	✗
[BPR12]	$[n, m = \mathsf{poly}(n), q = n^{\omega(1)}, t = \mathsf{poly}(n), \epsilon = n^{-\omega(1)}]_{\mathcal{R}_q}$ -Ring-LWE	Lattice-Based	✗
[AR16]	Strong Goldreich's OWF	Expander-Based	✗
[BIP+18]	Hash + Alternating Moduli	Alternating Moduli-Based	✗
[DJK25]	$[n, m = n^{\omega(1)}, \mu = n^{-o(1)}, t = n^{\omega(1)}, \epsilon = \mathsf{negl}(\lambda)]_2$ -Seeded-LPN	Code-Based	✓
This work	$[n, m = \mathsf{poly}(n), \mu = \frac{\mathsf{poly}(\log n)}{n}, t = \mathsf{poly}(n), \epsilon = \mathsf{negl}(n)]_{\mathcal{R}_q}$, -RingLPN	Code-Based	✗
This work	$[n, m = n^{1.01}, \mu = n^{-1/\log\log n}, t = \mathsf{poly}(n), \epsilon = \mathsf{negl}(n)]_2$ -SparseLPN	Code-Based	✓

Result 3: PRF in NC^1 from quasi-poly LPN Finally, as a bonus, we construct an NC^1 KH-PRF, albeit under the *quasi-polynomial hardness* of classical LPN assumption. This result is obtained by instantiating Theorem 1 (up to further refinement) with $n = \lambda^{o(1)}$, and matrix $\mathbf{M}$ chosen to be a random regular-sparse matrix.

Theorem 5 (Informal; see Corollary 3)**.** *Let λ be a security parameter. There exist KH-PRFs computable by NC^1 circuits assuming quasi-poly hardness of (classical) LPN for dimension $n = \lambda^{1/\log\log\lambda}$, number of samples $m = 2^{\log^{1.1}\lambda}$, noise rate $\mu = n^{-1/\log\log n}$.*

Remark 3. The non-trivial aspect of the above result is the combination of having logarithmic depth and being key-homomorphic (while relying on the classical LPN assumption). Indeed, one can either construct key-homomorphic PRFs with *super-logarithmic* depth using SparseLPN (namely, [DJK25]), or construct a *non-key-homomorphic* log-depth PRF from quasi-polynomial hardness of LPN by converting the complexity-leveraged weak PRF in [DJK25] to a strong PRF via [NR95].

Further remarks and discussions on our results. Our PRF security notion is the standard notion of polynomial security (i.e., any polynomial time adversary's advantage is bounded by a negligible function). Our PRFs support a super-polynomial input domain $\mathcal{X} = \{0,1\}^{\omega(\log\lambda)}$. Using generic domain-extension techniques, one can obtain an exponentially large input domain [Lev85,

BHKN13, DS15] while preserving the logarithmic depth; however, these techniques do not preserve key-homomorphism.

Conceptual Implications: Code-based vs. Lattice-based. We find the conceptual implications of this work intriguing. Recall that, historically, the development of code-based cryptography has arguably lagged behind that of lattice-based cryptography. In the context of PRFs, lattice-based constructions achieved:

1. NC^1 PRFs from Ring-LWE (i.e., [BPR12]), and
2. (Super-log-depth) KH-PRFs from LWE (i.e., [BLMR13]).

By obtaining NC^1 PRF from $\mathsf{RingLPN}$, and NC^1 KH-PRF from $\mathsf{SparseLPN}$, code-based cryptography appears to match, and in some sense even surpass, lattice-based cryptography in the context of PRFs.

Remark 4. It is also worth noting that our Results 1 and 3 yield puncturable PRFs in NC^1, whereas known lattice-based NC^1 PRFs are not puncturable.

Comparison with [DJK25]. We compare our work with [DJK25] along two dimensions.

- *Comparison of results.* The only NC^1 PRFs constructed in [DJK25] rely on the **quasi-polynomial hardness** of a new assumption, called seeded LPN. Our sparse LPN assumption is a **polynomial time** assumption (with polynomially bounded samples), while seeded LPN is a quasi-polynomial assumption with quasi-polynomial samples. Moreover, our construction relies on an assumption with a long history of study, rather than on a newly introduced one.
- *Technical comparison.* Our unified framework builds on two ideas from [DJK25]:
  - the recursive derandomization of a GGM-like construction; and
  - the use of sparse matrices.

  Beyond these inherited ideas, our framework introduces many new ingredients. Within this framework, we introduce a new construction template, including the use of a padding matrix, which makes our proof work for arbitrary matrix distributions satisfying the framework conditions. We also introduce the idea of using varying sparsities and varying noise rates, which are crucial for our classical-LPN-based instantiation. We also give a tighter security proof and a tighter depth analysis. These improvements are crucial for obtaining our results.

2 Technical Overview

Before we describe the main ideas behind our constructions we set up some minimal notation for the overview.

Notation: We consider inputs $x \in [\lambda]^k$, where λ is the security parameter and $k = \omega(1)$ is a small super-constant (e.g., $k = \log\log\log\lambda$). Let $n = O(\lambda)$ denote the dimension of the LPN assumption, and also the PRF output length. Let μ be the noise rate, with $\mu \gg 1/n$. We write $\ell := \log(1/\mu)$ for the number of random bits needed to sample a Bernoulli noise Ber_μ by taking a product (i.e., for $\mathbf{r} \in \{0,1\}^\ell$, $\mathsf{S\text{-}Ber}(\mathbf{r}) := \prod_{i\in\ell} r_i \in \{0,1\}$. $\mathsf{S\text{-}Ber}$ is extended by applying it on each ℓ-bit block separately, therefore shrinking input length by a factor of ℓ. Finally, our constructions are defined recursively with a parameter $\tau = \omega(1)$ (e.g., $\tau = \log\log\log\lambda$) that upper bounds the number of recursion layers.

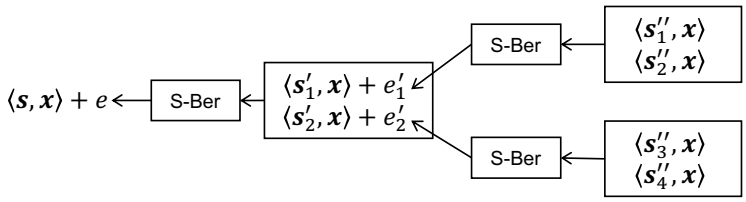


Figure 1: An illustration of weak PRFs from LPN via recursive derandomization, in a toy case of recursion depth $\tau = 2$, and $\ell = 2$, so $\mathsf{S\text{-}Ber}\colon \mathbb{F}_2^2 \mapsto \mathbb{F}_2$. The secret key is 7 vectors $\mathbf{s}, \mathbf{s}'_{1/2}, \mathbf{s}''_{1/2/3/4} \in \mathbb{F}_2^n$. On input $\mathbf{x} \in \mathbb{F}_2^n$, the output of the weak PRF F is $\langle \mathbf{s}, \mathbf{x}\rangle + e \in \mathbb{F}_2$, where $e := \mathsf{S\text{-}Ber}(\mathbf{r}') \in \mathbb{F}_2$ is obtained by applying $\mathsf{S\text{-}Ber}$ to the output of the preceding layer $\mathbf{r}' := (\langle \mathbf{s}'_i, \mathbf{x}\rangle + e'_i)_{i\in[2]} \in \mathbb{F}_2^2$, where $\{e'_i\}_{i\in[2]} := \mathsf{S\text{-}Ber}(\mathbf{r}'') \in \mathbb{F}_2^2$ is again obtained by applying $\mathsf{S\text{-}Ber}$ to the output of its preceding layer $\mathbf{r}'' := (\langle \mathbf{s}''_i, \mathbf{x}\rangle)_{i\in[4]} \in \mathbb{F}_2^4$. This is the base layer and no noise is added.

Since our constructions of low-depth PRFs extend and generalize ideas from [DJK25], it will be beneficial to start by first recalling their ideas.

Recursive Derandomization. A main idea of [DJK25] is that the LPN assumption almost directly gives us a weak PRF in NC^1. If one could materialize a function $F_{\mathbf{s}}(\mathbf{x}) = \langle \mathbf{s}, \mathbf{x}\rangle + e$ where $\mathbf{s} \in \mathbb{F}_2^n$ and e is a Bernoulli error chosen with probability μ, then this would form a weak PRF by the LPN assumption with noise rate μ. While the linear term $\langle \mathbf{s}, \mathbf{x}\rangle$ is NC^1 computable, it is less clear from which one derives e. Ding et al. [DJK25] suggest to generate the error recursively (illustrated in Figure 1), by calling functions $F_{\mathbf{s}'}(\mathbf{x})$ ℓ times to obtain ℓ bits, and deriving the noise by applying a Bernoulli sampling function $\mathsf{S\text{-}Ber}_\mu$.

We now describe the main ideas behind proving its security and NC^1 computability.

Efficiency:[7] The key observation for analyzing the required circuit depth is that each inner product of the form $\langle \mathbf{s}, \mathbf{x}\rangle$ can be computed in $\log n$ depth in parallel. Furthermore, each $\mathsf{S\text{-}Ber}$ requires $\log \ell$ depth to multiply ℓ bits. So the total evaluation depth is

$$O(\log n + \tau \cdot \log \ell).$$

As long as $\tau \cdot \log \ell = O(\log \lambda)$, we can evaluate the function in $O(\log \lambda)$ depth.

Security: The security proof relies on a certain resiliency property of the constructed function, such that any changes to the base layer (the right most layer in the picture 1) will not change the output with high probability. The intuition is that each $\mathsf{S\text{-}Ber}$ is insensitive to its input and outputs the value 0 with probability $1 - \mu$. Therefore, as we sequentially apply $\mathsf{S\text{-}Ber}$ for τ times, the value

[7] Note that it suffices to ensure logarithmic depth, since with bounded fan-in gates (in particular, fan-in 2), this automatically implies polynomial size.

at the final layer will hardly change. The idea in the proof is that one could start with a random string at the base (rightmost layer) and can replace $\{\langle \mathbf{x}, \mathbf{s}_i''' \rangle\}$ by a fresh random vector. This gives us almost the same function (in the sense that the output is unchanged), but for this function we can prove the (weak) pseudorandomness under LPN. Indeed, random vectors help us derive truly Bernoulli distributed error and appeal to the security of LPN inductively layer by layer to maintain this invariant.

PRF, recursively defined

Global Parameters: Max recursion layer τ. Let $T = O((k\ell)^\tau)$,

$$\mathbf{S}_{(k')} \in \mathbb{F}_2^{T \times n}, \text{ for } k' \in [k], \qquad \mathbf{A} = (\mathbf{A}_1, \ldots, \mathbf{A}_\lambda), \text{ where } \mathbf{A}_i \in \mathbb{F}_2^{n \times n} \text{ for } i \in [\lambda].$$

Notation: Denote $\mathbf{M}_{(i,j)} = \prod_{l=i}^{j} \mathbf{A}_{x_l}$.

For each input length $k' = 1, ..., k$, we define $\mathsf{F}_{(k')}$ as:

Function $\mathsf{F}_{(k')}(x \in [\lambda]^{k'}, \mathsf{layer})$:

1. If $\mathsf{layer} = \tau$: return $\mathbf{0} \in \mathbb{F}_2^n$.
2. For each input prefix $i = 1, ..., k'$, call $\mathsf{F}_{(i)}$ ℓ times (denoted by $\mathsf{F}_{(i)}^{\ell}$) and derive the noise vector:
$$\mathbf{e}_i \leftarrow \mathsf{S\text{-}Ber}_\mu(\mathbf{R}_i) \in \mathbb{F}_2^n, \quad \text{where } \mathbf{R}_i \leftarrow \mathsf{F}_{(i)}^{\ell}(x_{1,..,i}, \mathsf{layer}+1) \in \mathbb{F}_2^{\ell \times n},$$
3. Let $\mathbf{s} \in \mathbb{F}_2^n$ be an unused row of $\mathbf{S}_{(k')}$:
$$\begin{aligned} \mathbf{y} &\leftarrow (\cdots(\mathbf{s} \cdot \mathbf{A}_{x_1} + \mathbf{e}_1) \cdot \mathbf{A}_{x_2} + \mathbf{e}_2 \cdots) \cdot \mathbf{A}_{x_{k'}} + \mathbf{e}_{k'} \\ &= \mathbf{s} \cdot \mathbf{M}_{(1,k')} + \sum_{i=1}^{k'} \mathbf{e}_i \cdot \mathbf{M}_{(i+1,k')} \in \mathbb{F}_2^n, \end{aligned} \tag{1}$$
4. Return $\mathbf{y} \in \mathbb{F}_2^n$.

Figure 2: We define $\mathsf{PRF}(x) = \mathsf{F}_{(k)}(x, 0)$. Observe that a single evaluation of PRF makes a total of $T = O((k\ell)^\tau)$ recursive calls. Each call uses a vector $\mathbf{s}$ corresponding to a previously unused row of $\mathbf{S}_{(k')}$. Consequently, the ℓ invocations of $\mathsf{F}_{(i)}$ are performed with different $\mathbf{s}$, and hence return different values.

Extending to PRFs via a GGM-like approach. The above approach produces a weak PRF in NC^1 from LPN. To achieve a strong PRF, [DJK25] suggested to extend the above technique to a GGM-like construction. The observation is that we can view LPN samples as a "PRG" stretching a seed $\mathbf{s} \in \mathbb{F}_2^n$ to $\mathbf{s} \cdot \mathbf{A} + \mathbf{e} \in \mathbb{F}_2^m$. Then, we can repeatedly apply a PRG guided by an input-dependent

selection. The resulting construction is the recursive derandomization of the following:

$$\begin{aligned}\mathbf{y} &\leftarrow (\cdots(\mathbf{s}\cdot\mathbf{A}_{x_1}+\mathbf{e}_1)\cdot\mathbf{A}_{x_2}+\mathbf{e}_2\cdots)\cdot\mathbf{A}_{x_k}+\mathbf{e}_k \\ &= \mathbf{s}\prod_{i=1}^{k'}\mathbf{A}_{x_i}+\sum_{i=1}^{k'}\mathbf{e}_i\cdot\prod_{j=i+1}^{k'}\mathbf{A}_{x_j}\end{aligned} \tag{2}$$

More formally, on input $x\in[\lambda]^k$, the PRF outputs

$$\mathsf{PRF}(x;\ \mathbf{S},\mathbf{A}) \coloneqq \mathsf{F}_{(k)}(x,\ 0),\ \text{where } \mathsf{F}_{(k)} \text{ is recursively defined in Figure 2.}$$

A few remarks on how exactly the recursion is done. For a GGM-style proof to work, the error $\mathbf{e}_1$ should depend only on x_1, $\mathbf{e}_2$ should depend on x_1 and x_2, and any $\mathbf{e}_i$ should depend on the prefix of the input of length i. Therefore, in Figure 2, the function is defined for any input length $k'\le k$. For $i\in[k]$, it derives $\mathbf{e}_i$ by calling $\mathsf{F}_{(i)}$ ℓ times. The total depth of recursion is τ at which point the function returns $\mathbf{0}$ (thereby initiating the base layer with $\mathbf{0}$).

If each column of $\mathbf{A}$ is sparse, then

- For **security**, similar to weak PRFs, one can prove a resiliency property that the output is not sensitive to *any* changes in the base layer. Because the product of sparse matrices remain sparse. The matrix product $\prod\mathbf{A}$ is itself sufficiently sparse so that the computing $\sum\mathbf{e}\cdot\prod\mathbf{A}$ exhibits only local dependency on $\mathbf{e}$.
- For **key homomorphism**, since $\mathbf{e}$ and $\mathbf{A}$ are sparse, $\sum\mathbf{e}\prod\mathbf{A}\approx\mathbf{0}$. Therefore, the constructed PRF is close to a linear function on $\mathbf{s}$.

The biggest challenge is NC^1-computability. In general, computing the product of $k=\omega(1)$ matrices is unlikely to be NC^1. Therefore, [DJK25] only constructed an $\mathsf{NC}^{1+\epsilon}$ PRF from $\mathsf{SparseLPN}$ and had to settle the weaker notion of an encoded-input PRF (wherein the non-NC^1 computation is offloaded to a preprocessing phase). They then resort to a different approach to construct NC^1 PRFs, which necessitates a super-polynomial (ad hoc) assumption (called "Seeded-LPN").

In this work, we provide three new constructions of PRFs (two homomorphic, and one without homomorphism) from well-studied assumptions, circumventing the challenges faced by [DJK25]. Specifically, we manage to compute the above recursively defined function in NC^1 via three instantiations of the choice of the distributions of $\mathbf{S}$ and $\mathbf{A}$ (while admitting reductions to well-studied assumptions).

All three constructions are obtained as instantiations of a single unified framework presented in Section 4. This framework abstracts the properties of the matrix distribution that are needed for the construction to be both secure and computable in logarithmic depth (Section 4.1). Concretely, it gives a generic construction template, together with a security proof and an efficiency analysis: once the stated sufficient conditions are satisfied, the template immediately yields a secure PRF in NC^1.

In the rest of this technical overview, we first highlight some ideas behind this unified framework. We then explain how to instantiate it in three different ways to obtain our results (these instantiations also help motivate several design choices in the framework).

A Unified Framework. The PRF function is defined recursively (in Section 4.2), essentially as in Figure 2 with respect to parameters $\mathbf{S},\mathbf{A}$. (For now, we do not specify the exact distributions of

An equivalent recursive definition

Let global parameters and notation be as in Figure 2.
For each input length $k' = 1, ..., k$, we define $\mathsf{F}_{(k')}$ as:
Function $\mathsf{F}'_{(k')}(x \in [\lambda]^{k'}, \mathcal{I} \subseteq [n], \mathsf{layer})$:

1. If $\mathsf{layer} = \tau$: return $\mathbf{0} \in \mathbb{F}_2^n$.
2. For each input prefix $i = 1, ..., k'$:

$$\mathcal{I}_i \leftarrow \mathsf{RowSupp}(\overline{\mathbf{M}}_{(i+1,k')}), \text{ where } \overline{\mathbf{M}}_{(i+1,k')} \leftarrow \mathbf{M}_{(i+1,k')}|_{\mathcal{I}}$$

$$\overline{\mathbf{e}}_i \leftarrow \mathsf{S\text{-}Ber}_\mu(\mathbf{R}_i|_{\mathcal{I}_i}) \in \mathbb{F}_2^{|\mathcal{I}_i|}, \quad \text{where } \mathbf{R}_i|_{\mathcal{I}_i} \leftarrow (\mathsf{F}'_{(i)})^\ell(x_{1...i}, \mathcal{I}_i, \mathsf{layer}+1) \in \mathbb{F}_2^{\ell \times |\mathcal{I}_i|},$$

3. Let $\mathbf{s} \in \mathbb{F}_2^n$ be an unused row of $\mathbf{S}_{(k')}$: (letting $\overline{\mathbf{s}} \leftarrow \mathsf{ColSelect}(\mathbf{s}, \mathcal{I}_1)$)

$$\mathbf{y} = \overline{\mathbf{s}} \cdot \overline{\mathbf{M}}_{(1,k')} + \sum_{i=1}^{k'} \overline{\mathbf{e}}_i \cdot \overline{\mathbf{M}}_{(i+1,k')} \in \mathbb{F}_2^{|\mathcal{I}|}, \tag{3}$$

4. Return $\mathbf{y} \in \mathbb{F}_2^{|\mathcal{I}|}$.

Figure 3: An equivalent recursive definition.

these matrices, but only assume that $\mathbf{A}$ is a d-(column) sparse matrix, which is crucial to prove the resiliency property.) One technical but important modification we make is that before returning $\mathbf{y}$ after each execution of $\mathsf{F}_{(k')}$, we add a random padding $\mathbf{p}$ (treated as part of the key) to $\mathbf{y}$. This does not change the depth of the computation asymptotically, or, the key homomorphism properties of the output. However, this makes the analysis of the resiliency property *independent of* the distributions of the sparse matrices $\mathbf{A}$ and allows us to freely choose the distributions of $\mathbf{A}$ in order to squeeze the depth of PRF. For simplicity, from now on we ignore this additional padding in the remainder of this section.

Below we explain the idea in the depth analysis (Section 4.3 and Section 4.4), and refer to Section 4.5 for security analysis.

Dependency set via graph traversal. We consider an equivalent formulation, which we call PRF' that is simply an output restricted version of PRF defined before. Namely, this PRF' on input: $x \in [\lambda]^k$, and a support $\mathcal{I} \subseteq [n]$, outputs $\mathbf{y} = \mathsf{PRF}(x)|_{\mathcal{I}} \in \mathbb{F}_2^{|\mathcal{I}|}$, corresponding to indices in $\mathcal{I}$ as opposed to the entire output.

The reason we work with this augmented notation is that it helps us unravel the structure of the computation via a recursive framework that we can hope to analyze better. We claim that

$$\mathsf{PRF}(x) = \mathsf{PRF}'(x, \{1, \ldots, n\}) = \mathsf{F}'_{(k)}(x, \{1, \ldots, n\}, 0)$$

where F' is defined in Figure 3. Note that the first equality holds by definition of PRF'. The second equality holds by a closer inspection of Figure 3:

1. Since the input $\mathcal{I} \subseteq [n]$ tells us we only need to return $\mathbf{y}|_{\mathcal{I}}$, we do not need to compute $\mathbf{y}|_{[n]\setminus\mathcal{I}}$.
2. Observe that $\mathbf{y}|_{\mathcal{I}} = \mathbf{s} \cdot \mathbf{M}|_{\mathcal{I}} + \sum \mathbf{e} \cdot \mathbf{M}|_{\mathcal{I}}$, where $\mathbf{M}|_{\mathcal{I}}$ is the column restriction of $\mathbf{M}$ to $\mathcal{I}$. Let $\mathcal{I}' \subseteq [n]$ be the row support of $\mathbf{M}|_{\mathcal{I}}$ (i.e., the set of row indices that have a nonzero entry). We only need to know $\mathbf{e}|_{\mathcal{I}'} = \mathsf{S\text{-}Ber}(\mathbf{R})|_{\mathcal{I}'}$, which is actually equal to $\mathsf{S\text{-}Ber}(\mathbf{R}|_{\mathcal{I}'})$ since $\mathsf{S\text{-}Ber}$ operates on each column of $\mathbf{R}$ independently.
3. Each row $\mathbf{r}$ of $\mathbf{R}$ is returned by a recursive call. Since now we are only interested in entries in $\mathcal{I}'$, we make the recursive call with $\mathcal{I}'$. This yields a recursive closure.

Now that F' provides the desired functionality for us, we can compute each output bit $j \in [n]$ separately by calling $\mathsf{F}'_{(k)}(x, \{j\}, 0)$:

$$\mathsf{PRF}(x) = (\mathsf{F}'_{(k)}(x, \{j\}, 0))_{j \in [n]}.$$

Fix any $j \in [n]$ from now on. It suffices to analyze the circuit depth of $\mathsf{F}'_{(k)}(x, \{j\}, 0)$. Let the input to the PRF be $x \in [\lambda]^k$. Observe that throughout the evaluation of $\mathsf{PRF}(x)$, whenever a subroutine $\mathsf{F}'_{(k')}(\cdot, \cdot, \cdot)$ is invoked, its first argument is always $x_{1\ldots k'}$. Can we also 'predict' the second argument of $\mathsf{F}_{(k')}(\cdot, \cdot, \cdot)$? We hope that it only depends on k' (as hinted by our notation in Figure 3, $\mathsf{F}'_{(i)}$ is invoked with second argument $\mathcal{I}_i$ with a subscript that only depends on i), and independent of its actual caller $\mathsf{F}'_{(k'')}$ for some $k'' \geq k'$)?

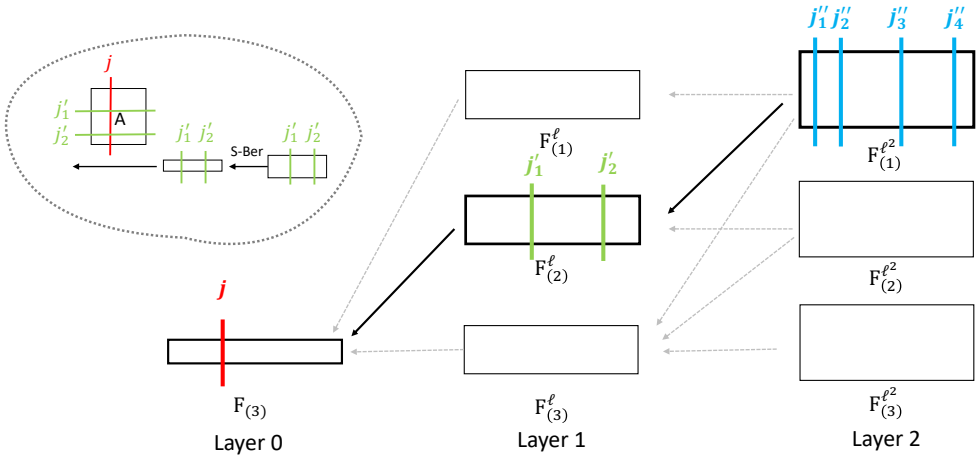


Figure 4: An illustration with $k = 3$. The dependency sequence $j \to j' \to j''$ can be viewed as a graph traversal on the layered graph induced by $\mathbf{A}_2, \mathbf{A}_3$.

We consider a slightly different interpretation of this, where these dependency sets $\mathcal{I}_1, \ldots, \mathcal{I}_k$ can be viewed as follows. We can start with $\mathcal{I}_k = \mathcal{I} = \{j\}$, and then construct $\mathcal{I}_i$ inductively from $\mathcal{I}_{i+1}$: $\mathcal{I}_i$ is defined as the set of all the nodes that are reachable by taking a 1-step walk from the starting set $\mathcal{I}_{i+1}$ on a graph whose adjacency matrix is given by $\mathbf{A}_{x_{i+1}}$.

In other words, the dependency set $\mathcal{I}_{k'}$ defined above can be viewed as the *reachability set* of a $(k - k')$-step walk on the layered graph induced by matrices $\mathbf{A}_{x_k}, \mathbf{A}_{x_{k-1}}, \ldots, \mathbf{A}_{x_1}$, where the starting node is j.

Note that the $\mathcal{I}_i$ constructed this way might be slightly different from the sets that are derived according to the row-support approach described previously. This is because multiplication of $\prod \mathbf{A}$ to form $\mathbf{M}_{(i,j)}$ can introduce accidental zero terms mod 2. Nevertheless, the sets formed by taking the walks as described above contain the sets formed using the row-support approach.

This graph-traversal-based interpretation gives us a mathematical closure guarantee for these sets that will be useful for analysis. Namely, one can show that throughout the evaluation of $\mathsf{PRF}(x)$, all of the recursive calls can always be performed by calling $\mathsf{F}'_{(k')}(\mathcal{I}_{k'})$ for $k' \leq k$ on these sets $\mathcal{I}_1, \ldots, \mathcal{I}_k$. In other words, there are fixed sets of dependency that are relevant across all recursive calls, and we do not need to worry about every coordinate.

With this understanding, the depth analysis considerably simplifies by exploiting the following facts:

- The maximum size of the sets $\mathcal{I}_1, \ldots, \mathcal{I}_k$ is small. For a starting set $\mathcal{I} = \{j\}$, each walk can introduce a multiplicative factor of at most d (the column sparsity of $\mathbf{A}$), and thus, the maximum size of these sets is bounded by $w = d^k$. We set it to be a subpolynomial by choosing d to be poly-logarithmic and k to be suitable super-constant.
- If one can precompute these sets, each computation involved in recursive calls to $\mathsf{F}'_{(k')}$ becomes a local computation that acts only on these sets $\mathcal{I}_{k'}$. The size of the set is bounded by $w = d^k$. We also note that each computation is of the form $\mathbf{e} \cdot \mathbf{M}$, corresponding to a linear function acting on at most w variables of $\mathbf{e}$. This can be computed in depth $O(\log w) = o(\log n)$ if the linear function that needs to be computed can be preprocessed. Furthermore, the Bernoulli noise derivation acts column-wise and is very local. Therefore this also does not pose a problem.

The total depth of the unrolled computation. Roughly speaking, unrolling the recursive definition is a $\tau = \omega(1)$ sequential computation of (1) the product $\mathbf{e} \cdot \mathbf{M}$ and (2) applying Bernoulli sample function. The Bernoulli sample function has very small depth. Therefore this also does not pose a problem. By applying the idea of dependency set, the product $\mathbf{e} \cdot \mathbf{M}$ can be done in $O(\log w)$ depth. A straightforward analysis would therefore give a total depth of $O(\tau \cdot \log w)$, which may fail to be $O(\log n)$ if the recursion depth τ is not sufficiently small. In Section 4.4, we give a more refined and tighter analysis showing that the overall depth is in fact

$$O(\min(k, \tau) \cdot \log w).$$

Observe that since $k = \omega(1)$ can be set to be arbitrarily small (therefore smaller than τ), we therefore only need to ensure $\log w = o(\log n)$. This makes the requirement on sparsity d independent of the recursion depth τ, which must be chosen sufficiently large in order to prove the resiliency property.

To briefly conclude the discussion so far, we can compute the PRF function in $O(\log \lambda)$ depth if

1. The graph-traversal reachability set $\mathcal{I}_1, \ldots, \mathcal{I}_k$ can be computed in $O(\log \lambda)$ depth.
2. The iterated matrix multiplication $\prod \mathbf{A}$ can be computed in $O(\log \lambda)$ depth.

Remark 5. *Graph-traversal reachability and iterated matrix multiplication are closely related, but they are not fully equivalent. The need for an explicit graph-traversal condition can be intuitively*

explained as follows. Consider the following toy computation trace:

$$\mathsf{S\text{-}Ber}(\mathsf{S\text{-}Ber}(\mathbf{E} \cdot \mathbf{A}_1) \cdot \mathbf{A}_2).$$

This is not the same as

$$\mathsf{S\text{-}Ber}(\mathsf{S\text{-}Ber}(\mathbf{E} \cdot \mathbf{A}_1 \cdot \mathbf{A}_2)),$$

because S-Ber *does not commute/associate with the surrounding matrix multiplications.*

With that big picture in mind, now we are ready to present how to instantiate $\mathbf{A}$ and $\mathbf{S}$, yielding our three new results.

Result 1: From (polynomial hardness of) SparseLPN with respect to a sublinear-depth expander graph. Consider an equivalence between a d-sparse matrix in $\mathbb{F}_2^{n \times m}$ and a d-regular expander graph. Namely, we consider d-left regular bipartite graphs where the left set of m vertices represent the columns of this matrix and the right set of n vertices represent the rows. Each left vertex in $[m]$ is connected to d vertices on the right corresponding to the d non-zero entries in the column. We now consider $\Gamma \colon [m] \to [n]^d$ as the neighbor function for this graph.

Random d-sparse matrices correspond to d-regular random expander graphs, whose neighbor function Γ can be computed in depth $O(\log m)$ and size $O(md \log n)$, which is in linear depth and exponential size (in its input length $\log m$). One way to implement this is to hardwire its truth table (a total of md values in $[n]$) and then perform a lookup based on input.

Our main observation is that if we consider expander graphs whose neighbor function is computable in sublinear depth, i.e., $o(\log m)$, barely smaller than random expanders, one can do a k-step graph traversal in depth $O(\log n)$ for some suitably chosen $k = \omega(1)$. Below we briefly explain the intuition and refer to Section 5.2 for the full details. Roughly speaking, one can start with a set $\mathcal{I} = \{j\}$ and apply $\Gamma(\mathcal{I}) = \{u_1, \ldots, u_d\}$ to obtain $\mathcal{I}_{k-1}$. Then, one can apply Γ to the elements in this set to obtain (upon renaming and possibly with duplicates) $\mathcal{I}_{k-2} = \{u_1, \ldots, u_{d^2}\}$. Continuing this way for all the k steps, we will eventually obtain a set of size d^k. So far this computation takes depth $k \cdot \mathsf{depth}(\Gamma)$. But the resulting vector has many duplicates and is not sorted. We need to accumulate it into the desired sparse vector. This can be done in $O(\log(d^k)) = O(\log \lambda)$ depth via a bin-style sorting algorithm.

To compute the iterated matrix product, we observe that $\prod_i \mathbf{A}_i$ over $\mathbb{F}_2$ is equivalent to counting the number of paths (modulo 2) between each pair of nodes, which then reduces to the graph traversal problem. In more detail, we again take a k-step walk starting from each node j and obtain a reachability set (that contains many duplicates). This time we count (modulo 2) the occurrence of every node $i \in [n]$ in the reachability set.

Thus, using SparseLPN with respect to sublinear-depth expander graphs, we can construct a key-homomorphic PRF in NC^1. In our setting, we need to work with $m = n^{1.01}$, $d = \mathsf{poly}(\log n)$, such explicit constructions are known [GUV09, KT22]. Indeed, their constructions are NC-computable, which translates into a depth of $\mathsf{poly}(\log \log n)$ here.

Finally, we remark that in the literature of explicit expanders, it is required that the neighbor function of the expander can be computed in polynomial time. However, for NC^1 PRFs, an exponential time expander will also work for us, as long as it has a sublinear depth, which is barely smaller than a random expander. In this sense, we believe the required amount of "structuredness" is very mild.

Result 2: From (polynomial hardness of) RingLPN. Our second idea is to resort to a distribution of matrices $\mathbf{A}$ for which iterated product of matrices is in logarithmic depth. For this, we let the coefficient matrix $\mathbf{A} = (\mathbf{A}_1, \ldots, \mathbf{A}_\lambda)$ be a random quasi-negacyclic matrix. More concretely, each $\mathbf{A}_i \in \mathbb{F}_q^{n \times n}$ corresponds to a random n-degree cyclotomic polynomial ring element $a_i \in \mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$[8] (where n is a power of two and q is an appropriately chosen prime with $O(\log n)$ bits). Then, iterated product of $\prod \mathbf{A}_{x_i}$ can be done in logarithmic depth, a property also used to construct logarithmic-depth PRFs from Ring-LWE [BPR12]. In more detail, through Chinese remainder representation, iterated product of ring elements $a_x = \prod a_{x_i}$ can be reduced to the problem of iterated integer multiplication, which is in $\mathsf{TC}^0 \subseteq \mathsf{NC}^1$ [BCH86, HAB02]. We then convert the result from the evaluation representation back to the coefficient representation. Recall that the conversion between them is a matrix-vector multiplication over $\mathbb{Z}_q$, which is in NC^1 because both integer multiplication and iterated integer addition are in NC^1.

Recall that $\mathbf{A}$ is required to be sparse. One can show that if a_i is a sparse polynomial, then the induced matrix $\mathbf{A}_i$ is also column-sparse. However, RingLPN with *sparse coefficients* is not known to be as hard as RingLPN. To fix this problem, we forgo key homomorphism completely and use a dual version of the template above. All we have to do is a renaming adjustment. We treat $\mathbf{A} = (\mathbf{A}_1, \ldots, \mathbf{A}_\lambda)$ as λ *secret* polynomials (in matrix representation) chosen from random sparse polynomials, and treat $\mathbf{S} \in \mathbb{F}_q^{T \times n}$ as the concatenation of T *public* random polynomials (in coefficient representation). This allows us to reduce the PRF security to RingLPN with *sparse secrets chosen from the noise distribution*. This variant can be proven to be as hard as RingLPN.

Now we rely on the connection between iterated matrix multiplication over $\mathbb{F}_q$ for $q = \lambda^{\omega(1)}$ and graph-traversal reachability set. Observe that if the multiplication is over $\mathbb{Z}$, then the reachability set can be simply derived from the non-zero entries in the output of iterated matrix product. Working over $\mathbb{F}_q$ may introduce accidental zero terms, but as long as $q = \lambda^{\omega(1)}$, the cancellation probability is negligible.

Result 3: Key Homomorphic PRF from quasi-polynomial hardness of LPN. We now explore the possibility of constructing NC^1 PRFs relying on LPN, without any additional structure, but assuming its quasi-polynomial hardness.

Recall that in our first result, if we are forced to use a random expander, then the only way to achieve in $O(\log \lambda)$ is to set $n = \lambda^{o(1)}$, e.g., $n(\lambda) = \lambda^{1/\log\log\lambda}$. In this case, $\log n = o(\log \lambda)$. The main obstacle is if this construction can be proven to be secure assuming standard LPN? Super-polynomial sparse LPN can be used to prove its security, but what about super-polynomial time standard LPN?

Our main idea is to carefully leverage recent reductions [BBTV25] that show hardness of sparse LPN under standard LPN. Roughly speaking, the reductions say that SparseLPN with d-sparse random coefficient, and noise rate μ is as hard as LPN with dimension $\approx d$, and the same noise rate μ. The reductions are also sample preserving.

However, we note that it poses an implicit restriction on the sparsity d. That is, the reduction is only meaningful for $d \gg 1/\mu$, otherwise the resulting assumption LPN is clearly false. However, on the other hand, our resilience analysis suggests that we should pick μ small enough such that $1/\mu \gg d$ or else we lose the resilience property. At this point, it would appear that we are stuck.

[8] Namely, the j-th row of $\mathbf{A}_i$ is $\mathsf{Vec}(X^{j-1} \cdot a_i)$, where $\mathsf{Vec}(\cdot)$ denotes the coefficient vector of a ring element. It then holds that $\mathsf{Vec}(s \cdot a_i) = \mathsf{Vec}(s) \cdot \mathbf{A}_i$.

Our idea is that instead of using the same sparsity d and noise rate μ for each of the k components of the input, we carefully craft a sequence of sparsity and error probabilities:

$$1/\mu_1 \gg d_2 \gg 1/\mu_2 \gg d_3 \gg 1/\mu_3 \gg \ldots \gg 1/\mu_{k-1} \gg d_k,$$

such that it simultaneously allows a meaningful reduction and enables the resilience property. In more detail, a meaningful reduction to quasi-poly LPN is satisfied since $d_i \gg 1/\mu_i$. A refined resiliency analysis requires that, each noise parameter must be larger than all later sparsity parameters. This is satisfied since $1/\mu_i \gg d_j$ for $j > i$.

Namely, the coefficient matrix is now $\mathbf{A} = (\mathbf{A}_{1,1}, \ldots, \mathbf{A}_{k,\lambda})$. The PRF construction is the recursive derandomization of

$$\begin{aligned} \mathsf{PRF}_{(k)}(x;\ \mathbf{s}) &\leftarrow (\cdots(\mathbf{s}\cdot\mathbf{A}_{1,x_1}+\mathbf{e}_1)\cdot\mathbf{A}_{2,x_2}+\mathbf{e}_2\cdots)\cdot\mathbf{A}_{k,x_k}+\mathbf{e}_k \\ &= \mathbf{s}\prod_{i=1}^{k}\mathbf{A}_{i,x_i}+\sum_{i=1}^{k}\mathbf{e}_i\cdot\prod_{j=i+1}^{k}\mathbf{A}_{j,x_j} \in \mathbb{F}_2^n, \end{aligned} \tag{4}$$

where

$$\mathbf{e}_i \leftarrow \mathsf{S\text{-}Ber}_{\mu_i}(\mathbf{R}_i) \in \mathbb{F}_2^n, \quad \text{and } \mathbf{A}_{i,\star} \in \mathbb{F}_2^{n\times n} \text{ has column sparsity } d_i.$$

Finally, we remark that the previous work of [DJK25] constructed weak PRFs in logarithmic depth from LPN. Assuming quasi-polynomial hardness of LPN, and choosing $n = \lambda^{o(1)}$, it can be converted into a synthesizer computable in $o(\log n)$ depth, which yields a PRF in $O(\log n)$ depth [NR95, NR99], but without key homomorphism. Our construction, on the other hand, preserves homomorphism. Curiously, this result gives us the first application of the reduction between LPN and sparse LPN to derive a cryptosystem based on standard LPN.

All of our constructions can be distilled into a unified construction template, and a unified analysis framework. In the technical sections, we present our results in this unified manner.

3 Preliminaries

Sets and distributions. We use $[n]$ to denote the set $\{1, \ldots, n\}$. We use $\mathsf{Funcs}[\mathcal{X}, \mathcal{Y}]$ to denote the set of all functions mapping $\mathcal{X}$ to $\mathcal{Y}$. 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 Ber_μ to denote the Bernoulli distribution with parameter μ, i.e., $\Pr[\mathsf{Ber}_\mu = 1] = \mu$, $\Pr[\mathsf{Ber}_\mu = 0] = 1-\mu$. Extending it over larger fields, we use $\mathsf{Ber}_\mu(\mathbb{F})$ to denote the Bernoulli distribution that returns 0 with probability $1-\mu$, and a uniformly random non-zero element of $\mathbb{F}\backslash\{0\}$ otherwise. By $X \sim D$ we denote that random variable X follows distribution D. For a distribution D, we use $s \leftarrow\!\$\ D$ to denote sampling an element s according to distribution D. For a finite set S, we use $s \leftarrow\!\$\ S$ to denote sampling s uniformly from finite set S. For a value x, we use $y \leftarrow x$ to denote assigning the value of x to y.

Lemma 1 (Collision of two Bernoulli random variables)**.** *Let $n, m \in \mathbb{N}$, $\mu > 0$. Let $(\mathbf{E}, \mathbf{E}')$ be an arbitrary (correlated) random variable supported on $\mathbb{F}_q^m \times \mathbb{F}_q^m$ such that both marginal distributions are m-dimensional Bernoulli distributions with parameter μ; that is, $\mathbf{E} \sim \mathsf{Ber}_\mu^m$ and $\mathbf{E}' \sim \mathsf{Ber}_\mu^m$. It holds that*

$$\Pr_{(\mathbf{E},\mathbf{E}')}[\mathbf{E} \neq \mathbf{E}'] \ \leq\ 2m\mu. \tag{5}$$

Proof. By a union bound over each dimension, $\Pr_{\mathbf{E}}\left[\mathbf{E} \neq \mathbf{0}\right] \leq m\,\mu$. The same bound also holds for $\mathbf{E}'$. By a union bound,

$$\Pr_{(\mathbf{E},\mathbf{E}')}\left[\mathbf{E} \neq \mathbf{0} \ \vee \ \mathbf{E}' \neq \mathbf{0}\right] \leq 2m\,\mu.$$

Finally, we conclude that

$$\begin{aligned}
\Pr_{(\mathbf{E},\mathbf{E}')}\left[\mathbf{E} \neq \mathbf{E}'\right] &= 1 - \Pr_{(\mathbf{E},\mathbf{E}')}\left[\mathbf{E} = \mathbf{E}'\right] \\
&\leq 1 - \Pr_{(\mathbf{E},\mathbf{E}')}\left[\mathbf{E} = \mathbf{E}' = \mathbf{0}\right] \\
&= \Pr_{(\mathbf{E},\mathbf{E}')}\left[\mathbf{E} \neq \mathbf{0} \ \vee \ \mathbf{E}' \neq \mathbf{0}\right] \\
&\leq 2m\,\mu.
\end{aligned}$$

□

Vectors and matrices. We adopt the following vector convention: when we left-multiply a vector $\mathbf{v}$, it is treated as a row vector; when we right multiply a vector $\mathbf{v}$, it is treated as a column vector. For a vector $\mathbf{v}$, we use $|\mathbf{v}|$ or $\mathsf{wt}(\mathbf{v})$ to denote the Hamming weight of $\mathbf{v}$. We denote the Hamming distance between two vectors $\mathbf{x}$ and $\mathbf{y}$ by $\Delta(\mathbf{x}, \mathbf{y})$.

For a matrix $\mathbf{M} \in \mathbb{F}^{n\times m}$, for $i \in [n]$, we use $\mathbf{M}[i,:] \in \mathbb{F}^{1\times m}$ to denote its i-th row in $\mathbb{F}^m$. If $i = 0$, $\mathbf{M}[i,:]$ is defined to be a zero vector in $\mathbb{F}^m$. Similarly, we use $\mathbf{M}[:,j] \in \mathbb{F}^{n\times 1}$ to denote its j-th column in $\mathbb{F}^n$. If $j = 0$, $\mathbf{M}[:,j]$ is defined to be a zero vector in $\mathbb{F}^n$. For matrices $\mathbf{A} \in \mathbb{F}^{n\times m_1}$ and $\mathbf{B} \in \mathbb{F}^{n\times m_2}$, write $\mathbf{A} \parallel \mathbf{B} \in \mathbb{F}^{n\times(m_1+m_2)}$ for the *column-wise* concatenation (appending the columns of $\mathbf{B}$ to $\mathbf{A}$). Similarly, for $\mathbf{A} \in \mathbb{F}^{n_1\times m}$ and $\mathbf{B} \in \mathbb{F}^{n_2\times m}$, write $\mathbf{A} /\!/ \mathbf{B} \in \mathbb{F}^{(n_1+n_2)\times m}$ for the *row-wise* concatenation (stacking the rows of $\mathbf{B}$ beneath $\mathbf{A}$). We defer more notation to Section 3.2.

Lists. We use an n-dimensional vector $\mathbf{v}$ to denote a list of n elements.

Iterators. An iterator Iter is a data structure supporting two operations, Init and Next. Given a matrix $\mathbf{A} \in \mathbb{F}^{T\times n}$, $\mathsf{Iter.Init}(\mathbf{A})$ initializes the iterator on $\mathbf{A}$ (with an internal cursor at the first row), and each call to $\mathsf{Iter.Next}()$ returns the next unused row of $\mathbf{A}$ (in increasing row index). Hence, once initialized, Iter can be invoked at most T times before it is exhausted.

Iterators in recursion. We use iterators in the recursive definitions of our functions. Whenever multiple recursive calls are made in parallel, we fix a canonical ordering of these calls, which induces a consistent assignment of iterator entries to the corresponding recursive calls.

Circuit model. We work in the standard Boolean-circuit model with $\{\mathrm{AND}, \mathrm{OR}, \mathrm{NOT}\}$ gates of bounded fan-in. We define NC circuits as follows.

Definition 1 (NC circuits)**.** For an integer $i > 0$, an NC^i circuit is a Boolean circuit $C_\lambda\colon \{0,1\}^\lambda \to \{0,1\}^{m(\lambda)}$ with $\{\mathrm{AND}, \mathrm{OR}, \mathrm{NOT}\}$ gates of bounded fan-in, depth $O(\log^i \lambda)$, size $\mathsf{poly}(\lambda)$. An NC circuit is a Boolean circuit $C_\lambda\colon \{0,1\}^\lambda \to \{0,1\}^{m(\lambda)}$ with $\{\mathrm{AND}, \mathrm{OR}, \mathrm{NOT}\}$ gates of bounded fan-in, depth $\mathsf{poly}(\log \lambda)$, size $\mathsf{poly}(\lambda)$.

Computational indistinguishability. We use the following simplified notation to ease presentation. The PRF security parameter throughout this paper is λ, and all algorithms (including the adversary) are implicitly given the security parameter λ in unary, and most other parameters are functions of λ. We omit λ when it is clear from the context. We say that a function $\epsilon = \epsilon(\lambda)$ is negligible if for every $c > 0$, it is smaller than $\frac{1}{\lambda^c}$ for a sufficiently large $\lambda > N_c$.

We consider adversaries interacting as part of probabilistic experiments called games. For an adversary $\mathcal{A}$ and two games $H_0(n)$, $H_1(n)$ with which it can interact, $\mathcal{A}$'s distinguishing advantage (as a function of n) is $\mathsf{Adv}_{H_0,H_1}(\mathcal{A}) := |\Pr[\mathcal{A} \textit{ accepts in } H_0] - \Pr[\mathcal{A} \text{ accepts in } H_1]|$. When we say that an algorithm is *efficient*, we mean that it is a probabilistic polynomial-time algorithm.

Definition 2 (Computational Indistinguishability)**.** We say that games (or distributions) H_0, H_1 are (t, ϵ)-computationally indistinguishable, and denote $H_0 \approx_{t,\epsilon} H_1$, if for any probabilistic distinguisher $\mathcal{A}$ running in time t, $\mathsf{Adv}_{H_0,H_1}(\mathcal{A}) \leq \epsilon$. We say H_0, H_1 are computationally indistinguishable, denoted by $H_0 \approx_{compt.} H_1$ if for every polynomial t, there exists a negligible ϵ, such that $H_0 \approx_{t,\epsilon} H_1$.

3.1 Pseudorandom Functions

In this section, we formally define pseudorandom functions [GGM86].

Definition 3 (PRFs [GGM86])**.** Let $q, t, \epsilon > 0$. A keyed function $F\colon \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ is called (q, t, ϵ)-pseudorandom if for any t-sized oracle-aided circuit $D^{(\cdot)}$ with at most q oracle gates, the distinguishing advantage

$$\left| \Pr_{k \leftarrow\$ \mathcal{K}} \left[D^{F_k(\cdot)} = 1 \right] - \Pr_{R \leftarrow\$ \mathsf{Funcs}[\mathcal{X},\mathcal{Y}]} \left[D^{R(\cdot)} = 1 \right] \right| \leq \epsilon.$$

Now, let $q(\cdot), t(\cdot), \epsilon(\cdot)$ all be functions of λ. A (q, t, ϵ)-PRF family is a sequence of keyed functions $\mathcal{F} = \{F_\lambda : \mathcal{K}_\lambda \times \mathcal{X}_\lambda \to \mathcal{Y}_\lambda\}_{\lambda \in \mathbb{N}}$ equipped with a polynomial time key sampling algorithm and a polynomial time evaluation algorithm where each F_λ is $(q(\lambda), t(\lambda), \epsilon(\lambda))$-pseudorandom. We say $\mathcal{F}$ is a secure PRFs family if for any polynomial $q(\cdot), t(\cdot)$, there exists a negligible function $\epsilon(\cdot)$, such that $\mathcal{F}$ is a (q, t, ϵ)-PRFs family. We say $\mathcal{F}$ is in NC^1 if each f_λ can be evaluated by an NC^1 circuit.

It will suffice to construct NC^1 PRFs for input domain $\mathcal{X} = \{0,1\}^{\omega(\log \lambda)}$ of slightly super-polynomial size. One can transform them to support input domain $\mathcal{X} = \{0,1\}^\lambda$ via domain extension techniques [Lev85, Lev87, BHKN13, BHKN19, DS15], and then obtain linear size PRFs [IKOS08, FLY22], while preserving logarithmic depth. We note that these transformations will not preserve key homomorphism.

Lemma 2 (**Domain Extension for NC^1 PRFs** [DS15])**.** *Let λ be a security parameter. Let q, t, ϵ, d all be functions of λ. Let $\mathcal{X} = \{0,1\}^{\omega(\log \lambda)}$ be an input domain. If there exists a (q, t, ϵ)-pseudorandom function $F\colon \{0,1\}^{k(\lambda)} \times \mathcal{X} \to \mathcal{Y}$ computable in depth $O(\log \lambda)$, then there exists an NC^1 $(\min(q, |\mathcal{X}|/2), t - \mathsf{poly}(\lambda), \epsilon + q \cdot 2^{-\Omega(\lambda)})$-pseudorandom function $F'\colon \mathcal{K}' \times \{0,1\}^\lambda \to \mathcal{Y}$.*

Key-homomorphic PRFs are a special family of PRFs that satisfy an additional algebraic property. Specifically, for a key-homomorphic PRF, the key space $\mathcal{K}$ and the range $\mathcal{Y}$ of the PRF exhibit certain group structures such that its evaluation on any fixed input $x \in \mathcal{X}$ is homomorphic with respect to these group structures. Key-homomorphic PRFs have many useful applications in symmetric cryptography and give rise to distributed PRFs, symmetric-key proxy re-encryption, and

updatable encryption [BLMR13, EPRS17, LT18, KLR19, BDGJ20, BEKS20], OT extension [Sch18], and secure aggregation [LLPT23]. To date, KH-PRFs (Definition 4) are only known to exist in the random oracle model. In the plain model, we only know how to achieve a relaxation of key-homomorphism called almost-key-homomorphism [BLMR13, DJK25].

Definition 4 (Approximate Key-Homomorphic PRFs [NPR99, BLMR13, DJK25])**.** Let $n \in \mathbb{N}$. Let $\gamma \in (0,1)$. Let $(\mathbb{G}, +)$ be a group. We say $T\colon \mathbb{G} \to \mathbb{F}^n$ is a γ-linear operator if for any $x_1, x_2 \in \mathbb{G}$, and any $\lambda_1, \lambda_2 \in \mathbb{Z}$, it holds that

$$T(\lambda_1 \cdot x_1 + \lambda_2 \cdot x_2) = \lambda_1 \cdot T(x_1) + \lambda_2 \cdot T(x_2) + \boldsymbol{\varepsilon},$$

where $\boldsymbol{\varepsilon} \in \mathbb{F}^n$ is a sparse vector such that $\mathsf{wt}(\boldsymbol{\varepsilon}) \leq \gamma n$. Let $F\colon \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ be a PRF (Definition 3). We say F is a γ-KH if for every $x \in \mathcal{X}$, $T_x := F(\cdot, x)\colon \mathcal{K} \to \mathcal{Y}$ is a γ-linear operator.

3.2 Additional Notation for Matrix Operations

In this section, we introduce some matrix operations that will be used.

Column selection. We define the column selection operator $\mathsf{ColSelect}$, which outputs a matrix formed by selecting columns of an input matrix, according to the column indices given by a list.

Definition 5 ($\mathsf{ColSelect}$)**.** Let $n, m, m' \in \mathbb{N}$. On input a matrix $\mathbf{A} \in \mathbb{F}^{n \times m}$ and a list $\mathbf{c} \in (0 \cup [m])^{m'}$, define

$$\mathsf{ColSelect}(\mathbf{A}, \mathbf{c}) := \mathbf{A}[:, \mathbf{c}_1] \,\|\, \cdots \,\|\, \mathbf{A}[:, \mathbf{c}_{m'}] \in \mathbb{F}^{n \times m'}.$$

Recall that $\mathbf{A}[:, 0]$ is defined to be a zero vector in $\mathbb{F}^n$.

Row selection. Similarly, we define row selection operator $\mathsf{RowSelect}$, which outputs a matrix formed by selecting rows of an input matrix, according to the row indices given by a list.

Definition 6 ($\mathsf{RowSelect}$)**.** Let $n, n', m \in \mathbb{N}$. On input a matrix $\mathbf{A} \in \mathbb{F}^{n \times m}$, and a list $\mathbf{r} \in (0 \cup [n])^{n'}$, define

$$\mathsf{RowSelect}(\mathbf{A}, \mathbf{r}) := \mathbf{A}[\mathbf{r}_1, :] \,/\!/\, \cdots \,/\!/\, \mathbf{A}[\mathbf{r}_{n'}, :] \in \mathbb{F}^{n' \times m}$$

Recall that $\mathbf{A}[0, :]$ is defined to be a zero vector in $\mathbb{F}^m$.

Support of a vector with padding. The operator Supp returns the list of (strictly increasing) indices of the nonzero positions of a vector and then pads with zeros at the end to a fixed output length.

Definition 7 (Supp)**.** Let $n, m \in \mathbb{N}$. For a padding parameter $w \in \mathbb{N}$, define $\mathsf{Supp}_w : \mathbb{F}^n \to (0 \cup [n])^w$. For a vector $\mathbf{v} \in \mathbb{F}^n$ with $t \leq w$ number of nonzero entries, let $(i_1, \ldots, i_t)$ be the strictly increasing list of indices of the nonzero entries of $\mathbf{v}$. We define

$$\mathsf{Supp}(\mathbf{v}; w) := (i_1, \ldots, i_t, \underbrace{0, \ldots, 0}_{w-t \text{ entries}}) \in (0 \cup [n])^w,$$

When the padding parameter w is omitted, we define

$$\mathsf{Supp}(\mathbf{v}) := (i_1, \ldots, i_t) \in [n]^t,$$

i.e., no padding is applied.

Row-support with padding. The operator $\mathsf{RowSupp}$ returns the list of (strictly increasing) indices of the nonzero row indices of a matrix and then pads with zeros at the end to a fixed output length.

Definition 8 ($\mathsf{RowSupp}$)**.** Let $n, m \in \mathbb{N}$. For a padding parameter $w \in \mathbb{N}$, define $\mathsf{RowSupp}_w : \mathbb{F}^{n\times m} \to (0 \cup [n])^w$. For a matrix $\mathbf{A} \in \mathbb{F}^{n\times m}$ with $t < w$ number of nonzero rows, let $(i_1, \ldots, i_t)$ be the strictly increasing list of indices of the nonzero rows of $\mathbf{A}$. We define

$$\mathsf{RowSupp}(\mathbf{A};\, w) \;\coloneqq\; (i_1, \ldots, i_t,\; \underbrace{0, \ldots, 0}_{w-t \text{ entries}}) \;\in\; (0 \cup [n])^w,$$

When the padding parameter w is omitted, we define

$$\mathsf{RowSupp}(\mathbf{A}) \;\coloneqq\; (i_1, \ldots, i_t) \;\in\; [n]^t,$$

i.e., no padding is applied.

Row pruning with padding. We then define row-pruning operator $\mathsf{RowPrune}$, which first removes all zero rows of the input matrix, then pads additional zero rows at the end to output a matrix with a fixed number of rows.

Definition 9 ($\mathsf{RowPrune}$)**.** Let $n, m, w \in \mathbb{N}$. For a padding parameter $w \in \mathbb{N}$, define $\mathsf{RowPrune}_w \colon \mathbb{F}^{n\times m} \to \mathbb{F}^{w\times m}$ as follows. On input a matrix $\mathbf{A} \in \mathbb{F}^{n\times m}$ with $t < w$ number of nonzero rows, define

$$\mathsf{RowPrune}(\mathbf{A};\, w) \;\coloneqq\; \mathsf{RowSelect}(\mathbf{A},\, \mathsf{RowSupp}(\mathbf{A};\, w))$$

Lemma 3. *Let all parameters be as in Definition 5-9. Let $\mathbb{F} = \mathbb{F}_q$. Every operation in Definition 5-9 can be done in $O(\log(n) + \log\log q)$ depth.*

Proof. See Appendix A.1. □

4 A Unified Framework: MVP

4.1 A Sufficient Condition for NC^1 PRFs

Definition 10 (Bernoulli sample function)**.** Let $\mu \in (0, 1), \ell > 0$. We say a function $\mathsf{S\text{-}Ber} \colon \mathbb{F}_q^\ell \to \mathbb{F}_q$ is a Bernoulli sample function if $\mathsf{S\text{-}Ber}_\mu(\mathbf{U}) \sim \mathsf{Ber}_\mu(\mathbb{F}_q)$ for $\mathbf{U} \leftarrow\!\$\, \mathbb{F}_q^\ell$.

We abuse notation and extend $\mathsf{S\text{-}Ber}_\mu$ to handle input $\mathbf{x} \in \mathbb{F}_q^{n\ell}$ by applying $\mathsf{S\text{-}Ber}_\mu$ separately on each ℓ-bit block $\mathbf{x}$, yielding an output in $\mathbb{F}_q^n$. More formally, on input $\mathbf{x} \in \mathbb{F}_q^{n\ell}$, define

$$\mathsf{S\text{-}Ber}(\mathbf{x};\, \mu) \coloneqq (\mathsf{S\text{-}Ber}_\mu(\mathbf{x}_{1\ldots\ell}),\, \ldots,\, \mathsf{S\text{-}Ber}_\mu(\mathbf{x}_{(n-1)\ell+1\ldots n\ell})) \in \mathbb{F}_q^n.$$

We abuse notation and extend $\mathsf{S\text{-}Ber}_\mu$ to handle inputs $\mathbf{X} \in \mathbb{F}_q^{n\ell \times m}$ by applying $\mathsf{S\text{-}Ber}_\mu$ separately on each column of $\mathbf{X}$, yielding an output in $\mathbb{F}_q^{n \times m}$. More formally, on input $\mathbf{X} \in \mathbb{F}_q^{n\ell \times m}$, define

$$\mathsf{S\text{-}Ber}(\mathbf{X};\, \mu) \coloneqq \mathsf{S\text{-}Ber}_\mu(\mathbf{X}[:,1]) \,\|\, \cdots \,\|\, \mathsf{S\text{-}Ber}_\mu(\mathbf{X}[:,m]) \in \mathbb{F}_q^{n \times m}.$$

Definition 11 (Configuration)**.** Let λ be a security parameter. A configuration Config specifies the following parameters.

1. Dimension n, modulus q.
2. Maximum recursion depth τ.
3. Number of input chunks k.
   (a) For $i \in [k]$, noise rate μ_i, along with a Bernoulli sample function $\mathsf{S\text{-}Ber}_{\mu_i} \colon \mathbb{F}_q^{\ell_i} \to \mathbb{F}_q$.
   (b) For $i \in [k]$, sparsity d_i, along with a distribution $\mathcal{M}_i$ for a d_i-sparse matrix $\mathbf{M}_i = (\mathbf{M}_{i,1}, \ldots, \mathbf{M}_{i,\lambda}) \in \mathbb{F}_q^{n \times \lambda n}$, i.e., each column of $\mathbf{M}_i$ has at most d_i nonzero entries.

We then introduce a basic primitive from graph theory that outputs the reachability set of a walk starting from a given node. This functionality enables us to compute all terms of the form $\mathbf{e} \cdot \prod \mathbf{A}$ in $o(\log n)$ depth via projecting both $\mathbf{e} \in \mathbb{F}_2^n$ and $\mathbf{A} \in \mathbb{F}^{n \times n}$ into a dimension (denoted by w) equal to the size of the reachability set (which is only $o(n)$),

Definition 12 (Iterated matrix multiplication)**.** We define a function $\mathsf{iMatMul}$ that takes as input adjacency matrices $\mathbf{M}_1, \ldots, \mathbf{M}_k$, where each $\mathbf{M}_i \in \mathbb{F}_q^{n \times n}$ is obtained by taking one of the $n \times n$ submatrices of a matrix sampled from $\mathcal{M}_i$. The output is $\prod_{i=1}^k \mathbf{M}_i \in \mathbb{F}_q^{n \times n}$.

Definition 13 (Reachability over a layered graph)**.** Consider a $(k+1)$-layer[9] directed simple graph with n[10] nodes per layer. Each node at layer i has an out-degree of d_i[11]. For each $i = 0, ..., k-1$, the edges from layer i to layer $i{+}1$ are specified by a neighbor function $\phi_i \colon [n] \to [n]^{d_i}$, such that

$$\text{there is an edge } (i,u) \;\rightsquigarrow\; (i{+}1, v) \quad \text{iff} \quad v \in \phi_{i+1}(u)$$

We define a function Reach that takes as input a starting node $u \in [n]$ and k adjacency matrices $\mathbf{M}_1, \ldots, \mathbf{M}_k$, where each $\mathbf{M}_i \in \mathbb{F}_q^{n \times n}$ (obtained by taking one of the $n \times n$ submatrix of a matrix sampled from $\mathcal{M}_i$) induces a neighbor function $\phi_i(j) \coloneqq \mathsf{Supp}(\mathsf{ColSelect}(\mathbf{M}_i, j))$. The output is a list of reachability sets $\{\mathbf{c}^{(i)}\}_{i=0}^k$, where each $\mathbf{c}^{(i)} \in [n]^{w_i}$ (with $w_i = \prod_{i=1}^j d_j$ as an upper bound of the reachability set) contains all nodes at layer i that are reachable from node u at layer 0. Each $\mathbf{c}^{(i)}$ is sorted in increasing order and padded with zeros at the end; e.g., $\mathbf{c}^{(i)} = (1, 4, 5, 0, \ldots, 0)$.

Remark 6. Reach and $\mathsf{iMatMul}$ are closely related, although not fully equivalent. Over $\mathbb{F}_2$, each entry of $\mathsf{iMatMul}$ can be viewed as counting, modulo 2, the number of paths between two nodes, whereas Reach determines whether there exists at least one path between the two nodes.

Definition 14 (Valid configuration)**.** We say a configuration is valid if

1. $k = \omega(1)$.

[9] k corresponds to number of PRF input chunks
[10] n corresponds to LPN dimension
[11] d_i corresponds to column sparsity of $\mathbf{M}_{i,\star}$

2. $\log k + \log\log q = O((\log\lambda)/\tau)$.
3. For $i \in [k]$, $\mathsf{S\text{-}Ber}_{\mu_i}$ can be computed in depth $O((\log\lambda)/\tau)$.
4. $\min(k,\tau)\cdot\sum_{i=1}^{k}\log d_i = O(\log\lambda)$.
5. $-\tau\cdot\log\left(\sum_{i=1}^{k} 2\mu_i(k\ell)^\tau\cdot\prod_{j=i+1}^{k} d_j\right) = \omega(k\log\lambda)$.
6. There exists a (randomized) circuit of depth $O(\log\lambda)$ that computes[12] $\mathsf{iMatMul}$ (for the distribution $\mathcal{M}$).
7. There exists a (randomized) circuit of depth $O(\log\lambda)$ that computes Reach (for the distribution $\mathcal{M}$).
8. For every $i \in [k]$, it holds that: For any polynomial $m = \mathsf{poly}(\lambda)$,

$$\mathbf{V}\cdot\mathbf{M}_i + \mathbf{E} \approx_{compt.} \mathbf{U},$$

where $\mathbf{V} \leftarrow\!\$\ \mathbb{F}_q^{m\times n}, \mathbf{M}_i \leftarrow\!\$\ \mathcal{M}_i \in \mathbb{F}_q^{n\times\lambda n}, \mathbf{E} \leftarrow\!\$\ \mathsf{Ber}_{\mu_i}^{m\times\lambda\cdot n}, \mathbf{U} \leftarrow\!\$\ \mathbb{F}_q^{m\times\lambda\cdot n}$.

- Furthermore, we say a configuration is key-homomorphic if for every $i \in [k]$,

$$(\mathbf{M}_i, \mathbf{V}\cdot\mathbf{M}_i + \mathbf{e}) \approx_{compt.} (\mathbf{M}_i, \mathbf{U}).$$

Namely, $\mathbf{M}_i$ is public.

Theorem 6 (Meta theorem)**.** *If there exists a valid (key-homomorphic) configuration, then there exist (key-homomorphic) PRFs in* NC^1.

In the rest of this section, we will prove Theorem 6 step by step.

4.2 Function Definition

In this section, we specify the input–output behavior of our PRF family. While the definition itself is constructive, evaluating it verbatim does not yield low circuit depth. The next section presents a depth-efficient evaluation of this function.

Definition 15 (Functionality of PRFs)**.** We define a keyed function

$$\mathsf{PRF}_{\mathbf{k}}\colon [\lambda]^k \to \mathbb{F}_q^n.$$

Parameters (to be specified).

1. Configuration $\mathsf{Config} = (n, q, k, \tau, (\mu_1,\ldots,\mu_k), (d_1,\ldots,d_k), (\mathsf{S\text{-}Ber}_{\mu_1},\ldots,\mathsf{S\text{-}Ber}_{\mu_k}), (\mathcal{M}_1,\ldots,\mathcal{M}_k))$.

Shorthand Notation.

- $\ell := \max(\ell_1,\ldots,\ell_k) = \ell_k$ denotes the maximum shrinking ratio due to applying $\mathsf{S\text{-}Ber}_{\mu_i}$.

[12] We say a (randomized) circuit C computes a function $f\colon \mathcal{X}\to\mathcal{Y}$ if for every $x\in\mathcal{X}$, $\Pr_r[C(x;r) = f(x)] \geq 2/3$.

Recursive Definition of F

Global Parameters: Max recursion depth τ, iterators, denoted by $\mathsf{Iter}_{(i)}$ for $i \in [k]$, and

$$(\mathbf{M}_{1,1}, \ldots, \mathbf{M}_{k,\lambda}), \text{ where } \mathbf{M}_{i,x} \in \mathbb{F}_q^{n \times n} \text{ for } i, x \in [k] \times [\lambda], \qquad \mathbf{V}_{(i)},\ \mathbf{P}_{(i)} \in \mathbb{F}_q^{T \times n}, \text{ for } i \in [k]$$

For every $k' = 1, \ldots, k$, define $\mathsf{F}_{(k')}$ as follows.

Function $\mathsf{F}_{(k')}(x \in [\lambda]^{k'},\ \mathsf{layer})$:

1. If $\mathsf{layer} = \tau$, return $\mathbf{0} \in \mathbb{F}_q^n$.
2. If $\mathsf{layer} = 0$, initialize an iterator $\mathsf{Iter}_{(k')}.\mathsf{Init}(\mathbf{V}_{(k')}, \mathbf{P}_{(k')})$.
3. $(\mathbf{v}, \mathbf{p}) \leftarrow \mathsf{Iter}_{(k')}.\mathsf{Next}() \in \mathbb{F}_q^n \times \mathbb{F}_q^n$.
4. For each input prefix $i \in [k']$:

$$\mathbf{e}_i \leftarrow \mathsf{S\text{-}Ber}_{\mu_i}(\mathbf{R}_i) \in \mathbb{F}_q^n, \quad \text{where} \quad \mathbf{R}_i \ \leftarrow \mathsf{F}_{(i)}^{\ell_i}(x_{1,..,i},\ \mathsf{layer}{+}1) \in \mathbb{F}_q^{\ell_i \times n}.$$

5. Compute:

$$\begin{aligned} \mathbf{y} \ &\leftarrow \ (\cdots(\mathbf{v} \cdot \mathbf{M}_{1,x_1} + \mathbf{e}_1) \cdot \mathbf{M}_{2,x_2} + \mathbf{e}_2 \cdots) \cdot \mathbf{M}_{k',x_{k'}} + \mathbf{e}_{k'} \\ &= \ \mathbf{v} \prod_{i=1}^{k'} \mathbf{M}_{i,x_i} + \sum_{i=1}^{k'} \mathbf{e}_i \cdot \prod_{j=i+1}^{k'} \mathbf{M}_{j,x_j} \in \mathbb{F}_q^n, \end{aligned}$$

6. Return $\mathbf{y} + \mathbf{p} \in \mathbb{F}_q^n$.

Figure 5: Functionality of PRFs, recursively defined.

- $T := (k\,\ell)^\tau$ denotes the (maximum) total number of recursive calls throughout the evaluation of PRF.

Function Key. $\mathbf{k} = (\mathbf{M}, \mathbf{V}, \mathbf{P})$, where

- $\mathbf{M} = (\mathbf{M}_1, \ldots, \mathbf{M}_k)$, where $\mathbf{M}_i = (\mathbf{M}_{i,1}, \ldots, \mathbf{M}_{i,\lambda}) \leftarrow\!\$\ \mathcal{M}_i$.
- $\mathbf{V} = (\mathbf{V}_{(1)}, \ldots, \mathbf{V}_{(k)})$, with $\mathbf{V}_{(i)} \leftarrow\!\$\ \mathbb{F}_q^{T \times n}$, for $i \in [k]$.
- $\mathbf{P} = (\mathbf{P}_{(1)}, \ldots, \mathbf{P}_{(k)})$, with $\mathbf{P}_{(i)} \leftarrow\!\$\ \mathbb{F}_q^{T \times n}$, for $i \in [k]$.

Functionality. On input $x \in [\lambda]^k$:

1. Parse $x = (x_1, \ldots, x_k)$ with $x_{k'} \in [\lambda]$ for $k' \in [k]$.

2. Invoke the recursive function F (Figure 5)

$$\mathbf{y} \leftarrow \mathsf{F}_{(k)}(x, 0) \in \mathbb{F}_q^n.$$

3. Return $\mathbf{y} \in \mathbb{F}_q^n$.

4.3 Evaluation Algorithm

In this section, we present a depth-efficient evaluation algorithm of the PRF functionality defined in Definition 15, which allows us to compute it in depth $O(\log \lambda)$.

Recursive Definition of Core G

Global Parameters: Max recursion depth τ, iterators, denoted by $\mathsf{Iter}_{(i)}$ for $i \in [k]$, and

$$\overline{\mathbf{M}}_{(l,r)} \in \mathbb{F}_q^{w_{l-1} \times w_r}, \text{ for } l \in [k],\ r \in [l, k], \qquad \overline{\mathbf{V}}_{(i)},\ \overline{\mathbf{P}}_{(i)} \in \mathbb{F}_q^{T \times w_0} \times \mathbb{F}_q^{T \times w_i}, \text{ for } i \in [k]$$

For every $k' = 1, \ldots, k$, define $\mathsf{G}_{(k')}$ as follows.

Function $\mathsf{G}_{(k')}(x \in [\lambda]^{k'}, \mathsf{layer})$:

1. If $\mathsf{layer} = \tau$, return $\mathbf{0} \in \mathbb{F}_q^{w_{k'}}$.
2. If $\mathsf{layer} = 0$, initialize an iterator $\mathsf{Iter}_{(k')}.\mathsf{Init}\big(\overline{\mathbf{V}}_{(k')}, \overline{\mathbf{P}}_{(k')}\big)$.
3. $(\overline{\mathbf{v}}, \overline{\mathbf{p}}) \leftarrow \mathsf{Iter}_{(k')}.\mathsf{Next}() \in \mathbb{F}_q^{w_0} \times \mathbb{F}_q^{w_{k'}}$.
4. For each input prefix $i \in [k']$:

$$\overline{\mathbf{e}}_i \leftarrow \mathsf{S\text{-}Ber}_{\mu_i}(\overline{\mathbf{R}}_i) \in \mathbb{F}_q^{w_i},$$
$$\text{where} \quad \overline{\mathbf{R}}_i \leftarrow \mathsf{G}_{(i)}^{\ell_i}(x_{1,..,i},\ \mathsf{layer}{+}1) \in \mathbb{F}_q^{\ell_i \times w_i}.$$

5. Compute

$$\overline{\mathbf{y}} \leftarrow \overline{\mathbf{v}} \cdot \overline{\mathbf{M}}_{(1,k')} + \sum_{i=1}^{k'} \overline{\mathbf{e}}_i \cdot \overline{\mathbf{M}}_{(i+1,k')} \in \mathbb{F}_q^{w_{k'}}.$$

6. Return $\overline{\mathbf{y}} + \overline{\mathbf{p}} \in \mathbb{F}_q^{w_{k'}}$.

Remark. All overlined objects denote submatrices/subvectors of dimension $w_\star$.

Figure 6: Evaluation of PRFs, recursively defined.

We are now ready to present our PRF construction, which is a depth-efficient algorithm for evaluating the functionality defined in Definition 15. The evaluation is given recursively, but with two crucial differences: (i) we precompute all products $\prod \mathbf{M}$ in advance; and (ii) at each recursive call we compute $\overline{\mathbf{e}} \cdot \overline{\mathbf{M}}$ (rather than $\mathbf{e} \cdot \mathbf{M}$), leveraging the precomputed reachability sets so that the computation and output live in dimension w instead of n.

Construction 1 (Meta construction)**.** We define an evaluation algorithm

$$\mathsf{Eval}_{\mathbf{k}} \colon [\lambda]^k \to \mathbb{F}_q^n.$$

Shorthand Notation.

- $\ell := \max(\ell_1, \ldots, \ell_k)$ denotes the maximum shrinking ratio due to applying $\mathsf{S\text{-}Ber}_{\mu_i}$.
- $d := \max(d_1, \ldots, d_k)$ denotes the maximum column sparsity of matrix $\mathbf{M}_{\star,\star}$.
- $T := (k\,\ell)^\tau$ denotes an upper bound on the total number of recursive calls throughout one evaluation of PRF.
- $w_i := \prod_{j=i+1}^{k} d_j$ denotes the maximum column sparsity of the product $\prod_{j=i+1}^{\tau} \mathbf{M}_{j,\star}$. Define $w_k := 1$.

Evaluation Algorithm. On input $x \in [\lambda]^k$:

1. Parse $x = (x_1, \ldots, x_k)$ with $x_{k'} \in [\lambda]$ for $k' \in [k]$.
2. (In parallel) pre-compute the iterated products

$$\mathbf{M}_{(l,r)} \leftarrow \mathsf{iMatMul}(\mathbf{M}_{l,x_l}, \ldots, \mathbf{M}_{r,x_r}) \in \mathbb{F}_q^{n\times n}, \qquad l \in [k],\ r \in [l,k].^{13}$$

3. For each output index $j \in [n]$ in parallel:
   (a) Compute reachability set from graph walking starting at node j:

$$(\mathcal{I}_k, \ldots, \mathcal{I}_0) \leftarrow \mathsf{Reach}(j,\ \mathbf{M}_{k,x_k},\ \mathbf{M}_{k-1,x_{k-1}}, \ldots, \mathbf{M}_{1,x_1})^{14}$$

   (b) Column selection and pruning for $l \in [k], r \in [l,k]$ (all in parallel):

$$\overline{\mathbf{M}}_{(l,r)} \leftarrow \mathsf{RowPrune}\left(\mathsf{ColSelect}\left(\mathbf{M}_{(l,r)}, \mathcal{I}_r\right);\ w_{l-1}\right) \in \mathbb{F}_q^{w_{l-1}\times w_r}, \qquad O(\log n)\text{ depth}$$

   (c) Column selection (all in parallel):

$$\overline{\mathbf{V}}_{(i)} \leftarrow \mathsf{ColSelect}\left(\mathbf{V}_{(i)}, \mathcal{I}_0;\ w_0\right) \in \mathbb{F}_q^{T\times w_0}, \quad \text{for } i \in [k] \quad O(\log n)\text{ depth}$$

$$\overline{\mathbf{P}}_{(i)} \leftarrow \mathsf{ColSelect}\left(\mathbf{P}_{(i)}, \mathcal{I}_i\,;w_i\right) \in \mathbb{F}_q^{T\times w_i}, \quad \text{for } i \in [k] \quad O(\log n)\text{ depth}$$

   (d) Invoke the recursive core G (Figure 6) with parameter $\overline{\mathbf{M}}_{(l,r)}, \overline{\mathbf{V}}_{(i)}, \overline{\mathbf{P}}_{(i)}$[15]:

$$\mathbf{y}_j \ \leftarrow\ \mathsf{G}_{(k)}(x, 0) \in \mathbb{F}_q.$$

4. Output $\mathbf{y} = (\mathbf{y}_1, \ldots, \mathbf{y}_n) \in \mathbb{F}_q^n$.

[13] We assume $\mathbf{M}_{(k+1,k)} := \mathbf{I}_n$ to simplify subsequent notation.
[14] Note that the subscripts are written in reverse order to simplify subsequent notation.
[15] Note that these matrices depend on j; we omit the subscript j for simplicity.

A Variant, F' (to facilitate analysis)

Global Parameters: Max recursion depth τ, iterators, denoted by $\mathsf{Iter}_{(i)}$ for $i \in [k]$, and

$$(\mathbf{M}_{1,1}, \ldots, \mathbf{M}_{k,\lambda}), \text{ where } \mathbf{M}_{i,x} \in \mathbb{F}_q^{n \times n} \text{ for } i, x \in [k] \times [\lambda], \qquad \mathbf{V}_{(i)},\ \mathbf{P}_{(i)} \in \mathbb{F}_q^{T \times n}, \text{ for } i \in [k]$$

For every $k' = 1, \ldots, k$, define $\mathsf{F}'_{(k')}$ as follows.

Function $\mathsf{F}'_{(k')}(x \in [\lambda]^{k'},\ \mathcal{I} \subseteq [n],\ \mathsf{layer})$:

1. If $\mathsf{layer} = \tau$, return $\mathbf{0} \in \mathbb{F}_q^{|\mathcal{I}|}$.
2. If $\mathsf{layer} = 0$, initialize an iterator $\mathsf{Iter}_{(k')}.\mathsf{Init}(\mathbf{V}_{(k')}, \mathbf{P}_{(k')})$.
3. $(\mathbf{v}, \mathbf{p}) \leftarrow \mathsf{Iter}_{(k')}.\mathsf{Next}() \in \mathbb{F}_q^n \times \mathbb{F}_q^n$.
4. For each $i = 0, \ldots, k'$,
$$\mathcal{I}_i \leftarrow \mathsf{RowSupp}(\mathsf{ColSelect}(\mathbf{A}_{(i+1,k')}, \mathcal{I}))$$
5. For each input prefix $i \in [k']$:
$$\overline{\mathbf{e}}_i \leftarrow \mathsf{S\text{-}Ber}_{\mu_i}(\overline{\mathbf{R}}_i) \in \mathbb{F}_q^{|\mathcal{I}_i|}, \quad \text{where} \quad \overline{\mathbf{R}}_i \leftarrow \mathsf{F}'^{\ell_i}_{(i)}(x_{1,..,i},\ \mathcal{I}_i,\ \mathsf{layer}{+}1) \in \mathbb{F}_q^{\ell_i \times |\mathcal{I}_i|}.$$
6. Compute
$$\overline{\mathbf{y}} \leftarrow \overline{\mathbf{v}} \cdot \overline{\mathbf{M}}_{(1,k')} + \sum_{i=1}^{k'} \overline{\mathbf{e}}_i \cdot \overline{\mathbf{M}}_{(i+1,k')} \in \mathbb{F}_q^{|\mathcal{I}|}.$$
where
$$\overline{\mathbf{v}} \leftarrow \mathsf{ColSelect}(\mathbf{v}, \mathcal{I}_0) \in \mathbb{F}_q^{|\mathcal{I}_0|}$$
7. Return $\overline{\mathbf{y}} + \overline{\mathbf{p}} \in \mathbb{F}_q^{|\mathcal{I}|}$, where $\overline{\mathbf{p}} \leftarrow \mathsf{ColSelect}(\mathbf{p}, \mathcal{I}) \in \mathbb{F}_q^{|\mathcal{I}|}$.

Remark. All overlined objects denote submatrices/subvectors of dimension $|\mathcal{I}_\star|$.

Figure 7: A Variant, F' (to facilitate analysis)

Theorem 7. Eval *computes* PRF *(Definition 15). That is, for every* $\mathbf{k} \in \mathcal{K}, x \in \mathcal{X}$, $\mathsf{Eval}_{\mathbf{k}}(x) = \mathsf{PRF}_{\mathbf{k}}(x)$.

Proof. Fix any PRF secret key $\mathbf{k}$. We introduce a sequence of hybrids as follows.

- $\mathsf{PRF}_{\mathbf{k}}(\cdot, \cdot)$ (output restriction): This overloads the PRF defined in Definition 15. In addition to x, it takes an index set $\mathcal{I} \subseteq [n]$ and returns the restriction $\mathsf{PRF}(x)|_{\mathcal{I}} \in \mathbb{F}_2^{|\mathcal{I}|}$, instead of the

full vector $\mathsf{PRF}(x) \in \mathbb{F}_2^n$. In particular, $\mathsf{PRF}(x) = \mathsf{PRF}(x, \{1, \ldots, n\})$.[16]

- $\mathsf{PRF}'_{\mathbf{k}}(\cdot, \cdot)$: This is the same as the above PRF except that instead of calling F and selecting the output based on $\mathcal{I}$ via $\mathsf{PRF}(x, \mathcal{I}) \coloneqq \mathsf{ColSelect}(\mathsf{F}(x, 0), \mathcal{I})$, it directly calls F' (Figure 7) to output $\mathsf{PRF}'(x, \mathcal{I}) \coloneqq \mathsf{F}'(x, \mathcal{I}, 0)$.
- $\mathsf{Eval}_{\mathbf{k}}(\cdot)$: This is the our evaluation algorithm (Construction 1).

It remains to show that $\mathsf{PRF}_{\mathbf{k}} \equiv \mathsf{PRF}'_{\mathbf{k}}$ and $\mathsf{PRF}'_{\mathbf{k}}(\cdot, \{1, \ldots, n\}) \equiv \mathsf{Eval}_{\mathbf{k}}(\cdot)$. See Appendix A.2. □

4.4 Efficiency Analysis

Theorem 8. Eval *is in* $O(\log \lambda)$ *depth if*

1. $\log k + \log \log q = O((\log \lambda)/\tau)$
2. *For* $i \in [k]$, $\mathsf{S\text{-}Ber}_{\mu_i}$ *can be computed in depth* $O((\log \lambda)/\tau)$.
3. $\min(k, \tau) \cdot \sum_{i=1}^{k} \log d_i = O(\log \lambda)$.
4. $\mathsf{iMatMul}$ *and* Reach *can be computed in* $O(\log \lambda)$ *depth.*

Proof. We examine the depth required for each step.

1. The iterated matrix multiplication in step 2 is done by first selecting all $\mathbf{M}_{i,x_i}$ based on x_i, which can be done in depth $O(\log \lambda)$. Then $\mathsf{iMatMul}$ can be done in $O(\log \lambda)$ depth since Config is valid. The same reasoning also applies to step 3(a).
2. By Lemma 3, $\mathsf{ColSelect}$, $\mathsf{RowPrune}$ operations can be computed in depth $O(\log n) = O(\log \lambda)$.
3. It remains to analyze the circuit depth needed to evaluate the recursion G. Observe that the *recursion topology* is fixed and independent of the PRF input x and key: it is a rooted tree with depth τ and fanout $k\ell_i$ at tree-depth i. Tree root produces the PRF output. Thus the circuit evaluation amounts to unrolling this tree and computing bottom–up from the leaves to the root. At each internal node at tree-depth i, given the output of its $k\ell_i$ number of children, it first apply $\mathsf{S\text{-}Ber}$, which takes depth $dep(\mathsf{S\text{-}Ber})$, then compute the vector matrix multiplication at dimension of at most w_0, which takes depth $\log w_0 = \sum_{i=1}^{k} \log d_i$. A trivial analysis suggest a total depth of $O(\tau \cdot (dep(\mathsf{S\text{-}Ber}) + \log w_0))$. Below we do a tighter analysis to show that the total depth can be in $O(\tau \cdot dep(\mathsf{S\text{-}Ber}) + \log w_0)$ (i.e., an additive term for all the inner products), which is $O(\log \lambda)$ by plugging into $dep(\mathsf{S\text{-}Ber}) = O(\log \lambda/\tau)$, and $\sum_{i=1}^{k} \log d_i = O(\log \lambda)$.

Lemma 4 (Depth of recursion)**.** $\mathsf{G}_{(k)}(x, 0)$ *can be computed by a circuit of depth*

$$O\left(\tau \cdot (dep(\mathsf{S\text{-}Ber}) + \log k + \log \log q) + \min(k, \tau) \cdot \sum_{i=1}^{k} \log d_i\right).$$

[16] We abuse notation and write PRF for both interfaces; the intended meaning is clear from whether a second input is present.

G': A callee-delegated version of G

Global Parameters: Max recursion depth τ, iterators, denoted by $\mathsf{Iter}_{(i)}$ for $i \in [k]$, and

$$\overline{\mathbf{M}}_{(l,r)} \in \mathbb{F}_q^{w_{l-1} \times w_r}, \text{ for } l \in [k],\ r \in [l,k], \qquad \overline{\mathbf{V}}_{(i)},\ \overline{\mathbf{P}}_{(i)} \in \mathbb{F}_q^{T \times w_0} \times \mathbb{F}_q^{T \times w_i}, \text{ for } i \in [k]$$

For every $k' = 1, \ldots, k$, define $\mathsf{G}_{(k')}$ as follows.

Function $\mathsf{G}_{(k')}(x \in [\lambda]^{k'},\ \mathsf{layer})$:

1. If $\mathsf{layer} = \tau$, return $\mathbf{0} \in \mathbb{F}_q^{w_{k'}}$.
2. If $\mathsf{layer} = 0$, initialize an iterator $\mathsf{Iter}_{(k')}.\mathsf{Init}(\overline{\mathbf{V}}_{(k')}, \overline{\mathbf{P}}_{(k')})$.
3. $(\overline{\mathbf{v}}, \overline{\mathbf{p}}) \leftarrow \mathsf{Iter}_{(k')}.\mathsf{Next}() \in \mathbb{F}_q^{w_0} \times \mathbb{F}_q^{w_{k'}}$.
4. For each input prefix $i \in [k']$:
$$\overline{\mathbf{e}}'_i \leftarrow \mathsf{G}'^{\ell_i}_{(i)}(x_{1,..,i},\ \mathsf{layer}{+}1) \in \mathbb{F}_q^{w_i}$$
5. Compute
$$\overline{\mathbf{y}} \leftarrow \overline{\mathbf{v}} \cdot \overline{\mathbf{M}}_{(1,k')} + \sum_{i \in [k']} \overline{\mathbf{e}}'_i + \overline{\mathbf{p}} \in \mathbb{F}_q^{w_{k'}}.$$
6. If $\mathsf{layer} = 0$: return $\overline{\mathbf{y}} \in \mathbb{F}_q$.
7. Collect $\overline{\mathbf{y}}$ from its $\ell_{k'}$ siblings to form $\overline{\mathbf{R}}_{k'} \in \mathbb{F}_q^{\ell_{k'} \times w_{k'}}$. Compute
$$\overline{\mathbf{e}}_{k'} \leftarrow \mathsf{S\text{-}Ber}_{\mu_{k'}}(\overline{\mathbf{R}}_{k'}) \in \mathbb{F}_q^{w_{k'}}$$
8. Denote their caller by $\mathsf{G}_{k''}$. If $k'' = k'$, let $\overline{\mathbf{e}}'_{k'} \leftarrow \overline{\mathbf{e}}_{k'}$. Otherwise (i.e., $k'' > k'$), compute
$$\overline{\mathbf{e}}'_{k'} \leftarrow \overline{\mathbf{e}}_{k'} \cdot \overline{\mathbf{M}}_{(k'+1,k'')} \in \mathbb{F}_q^{w_{k''}} \quad O(\log w_{k'} + \log\log q) \text{ depth}$$
9. Return $\overline{\mathbf{e}}'_{k'} \in \mathbb{F}_q^{w_{k''}}$.

Figure 8: G': A callee-delegated version of G, in which the computation of $\overline{\mathbf{e}} \cdot \overline{\mathbf{M}}$ is delegated by the callers to the callees.

Proof. Unrolling the recursion forms a tree with tree-depth τ and fanout at most $k\ell$. First observe that all the products of the form $\overline{\mathbf{v}} \cdot \overline{\mathbf{M}}$ can be computed in parallel, which takes depth $O(\log w_0 + \log\log q)$. We then introduce an equivalent version G' (Figure 8), which delegates the product of the form $\overline{\mathbf{e}} \cdot \overline{\mathbf{M}}$ from the caller to its callees.

Definition 16 (Path)**.** A path is a length-τ non-decreasing sequence $s = (s_1, \ldots, s_\tau)$, where

$$1 \le s_1 \le s_2 \le \cdots \le s_\tau = k,$$

which corresponds to a recursion path $\mathsf{G}'_{(s_\tau)} \leftarrow \mathsf{G}'_{(s_{\tau-1})} \leftarrow \cdots \mathsf{G}'_{(s_1)}$. We define the length of a path s,

$$len(s) \coloneqq \sum_{i=i}^{\tau} d(\mathsf{G}'_{(s_i)}),$$

where $d(\mathsf{G}'_{(s_i)})$ denotes the circuit depth to perform the computation steps in $\mathsf{G}'_{(s_i)}$ (i.e., the return value of its recursive calls are treated as its input).

The total circuit depth to compute $\mathsf{G}'_{(k)}(x, 0)$ is

$$O(\log w_0 + \log\log q + \max_s len(s))$$

It suffices to prove the following lemma.

Lemma 5. *For every path s,*

$$len(s) = O\left(\tau \cdot (dep(\mathsf{S\text{-}Ber}) + \log k + \log\log q) + \min(k, \tau) \cdot \sum_{i=1}^{k} \log d_i\right).$$

Proof. For $k' \in [k]$, we study $d(\mathsf{F}_{(k')})$. Computation happens at step 5, 7, 8 of Figure 8.

1. Step 5 takes depth $O(\log k' + \log\log q)$ for adding $O(k)$ vectors in $\mathbb{F}_q$ (recall the $\overline{\mathbf{v}} \cdot \overline{\mathbf{M}}$ is precomputed).
2. Step 7 takes depth $dep(\mathsf{S\text{-}Ber})$.
3. Step 8 takes depth $O(\log w_{k'} + \log\log q)$ to compute $\overline{\mathbf{e}} \cdot \overline{\mathbf{M}}$ if it is called by $d(\mathsf{G}'_{(k'')})$ for $k'' > k$. Otherwise, step 8 takes 0 depth.

Therefore, we have

$$d(\mathsf{G}'_{(k')}) = \begin{cases} O(dep(\mathsf{S\text{-}Ber}) + \log w_{k'} + \log k' + \log\log q) & \text{, if called by } k'' > k \\ O(dep(\mathsf{S\text{-}Ber}) + \log k' + \log\log q) & \text{, if called by } k'' = k \end{cases}$$

Fix any path $s = (s_1, \ldots, s_\tau)$, in order to study

$$len(s) = \sum_{i=1}^{\tau} d(\mathsf{G}'_{(s_i)}),$$

we consider the contribution of each term in $d(\mathsf{G}'_{(s_i)})$:

1. The term $dep(\mathsf{S\text{-}Ber}, \log k', \log\log q$ will occur τ times in $len(s)$. They will contribute a total depth of
$$O(\tau \cdot (dep(\mathsf{S\text{-}Ber}) + \log k + \log\log q))$$
2. For every $k' \in [k]$, the term $\log w_{k'}$ ($\leq \log w_1$) will only occur at most once in $len(s)$. This is because s_i are non-decreasing, so there only exists at most one index $i \in [\tau]$, such that $s_i = k'$ and $s_{i-1} > k'$ (s_{i-1} is the caller of s_i), which will result in the term $\log w_{k'}$. On the other

hand, at most τ terms of $\log w_{k'}$ will contribute since $i \le \tau$ so the event $s_i > s_{i-1}$ can happen at most τ times. Therefore, the total contribution is

$$O(\min(k,\tau)\cdot \log w_1) = O\left(\min(k,\tau)\cdot \sum_{i=1}^{k} \log d_i\right)$$

Summing the above yields the lemma.

□

□

□

4.5 Security Analysis

Theorem 9. *Let $t, \epsilon_1, \epsilon_2 > 0$, $Q \le \lambda^k$. If for every $i \in [k]$,*

$$\mathbf{V}\cdot\mathbf{M}_i + \mathbf{E} \approx_{t,\epsilon} \mathbf{U},$$

where $\mathbf{V} \leftarrow\!\$\ \mathbb{F}_q^{Q\times n}, \mathbf{M}_i \leftarrow\!\$\ \mathcal{M}_i \in \mathbb{F}_q^{n\times \lambda n}$, $\mathbf{E} \leftarrow\!\$\ \mathsf{Ber}_{\mu_i}^{Q\times \lambda\cdot n}, \mathbf{U} \leftarrow\!\$\ \mathbb{F}_q^{Q\times \lambda\cdot n}$. Then, $\mathsf{PRF}_{\mathbf{k}}\colon \{0,1\}^{k\log\lambda} \to \{0,1\}^n$ defined in Definition 15 is a

$$(Q, t - \mathsf{poly}(\lambda), \tau k^2\cdot\epsilon + \lambda^k\cdot n\cdot\epsilon')$$

-pseudorandom function (Definition 3), where

$$\epsilon' \le \left(\sum_{i=1}^{k} 2\mu_i(k\ell)^\tau \cdot \prod_{j=i+1}^{k} d_j\right)^\tau$$

Proof. We define hybrid games as follows.

Hybrid H_0. This is the real PRF game: the attacker has oracle access to $\mathsf{PRF}_{\mathbf{k}}(\cdot)$ for $\mathbf{k} \leftarrow\!\$\ \mathcal{K}$.

Hybrid H_1. This is the same as Hybrid H_0 except that the attacker has oracle access to $\widetilde{\mathsf{PRF}}_{\mathbf{k}}(\cdot)$, which is defined the same as $\mathsf{PRF}_{\mathbf{k}}$, except that instead of invoking F, it will invoke $\widetilde{\mathsf{F}}$ defined in Figure 9.

Hybrid H_2. This is the same as Hybrid H_1 except that instead of invoking $\widetilde{\mathsf{F}}$, $\widetilde{\mathsf{PRF}}_{\mathbf{k}}$ will invoke $\widetilde{\mathsf{G}}$ defined in Figure 10 (invoke for each output bit as we did in Construction 1).

Hybrid H_3. This is the ideal PRF game: the attacker has oracle access to a random function $R(\cdot)$.

As a corollary of Theorem 7, it follows that $H_1 \equiv H_2$, since they are just two different ways to compute the same function. It suffices to show that $H_0 \approx_{stat.} H_2$ (resilience property), $H_1 \approx_{compt.} H_3$ (GGM-like argument). See Appendix A.3. □

Hybrid H_1: $\widetilde{\mathsf{F}}$

Global Parameters: Max recursion depth τ, iterators, denoted by $\mathsf{Iter}_{(i)}$ for $i \in [k]$, and

$$(\mathbf{M}_{1,1}, \ldots, \mathbf{M}_{k,\lambda}), \text{ where } \mathbf{M}_{i,x} \in \mathbb{F}_q^{n \times n} \text{ for } i, x \in [k] \times [\lambda], \qquad \mathbf{V}_{(i)},\ \mathbf{P}_{(i)} \in \mathbb{F}_q^{T \times n}, \text{ for } i \in [k]$$

For each input length $i \in [k]$.

$$R_{(i)} = (R_1, ..., R_T), \text{ where each } R_j \text{ is a random function } R_j \colon [\lambda]^i \to \mathbb{F}_q^n$$

For every $k' = 1, \ldots, k$, define $\widetilde{\mathsf{F}}_{(k')}$ as follows.

Function $\widetilde{\mathsf{F}}_{(k')}(x \in [\lambda]^{k'},\ \mathsf{layer})$:

1. If $\mathsf{layer} = \tau$, return $R(x) \in \mathbb{F}_q^n$, where $R \leftarrow \mathsf{Iter}'_{(k')}.\mathsf{Next}()$.
2. If $\mathsf{layer} = 0$, initialize an iterator $\mathsf{Iter}_{(k')}.\mathsf{Init}(\mathbf{V}_{(k')}, \mathbf{P}_{(k')}, R_{(k')})$.
3. $(\mathbf{v},\ \mathbf{p}) \leftarrow \mathsf{Iter}_{(k')}.\mathsf{Next}() \in \mathbb{F}_q^n \times \mathbb{F}_q^n$.
4. For each input prefix $i \in [k']$:

$$\mathbf{e}_i \leftarrow \mathsf{S\text{-}Ber}_{\mu_i}(\mathbf{R}_i) \in \mathbb{F}_q^n, \quad \text{where} \quad \mathbf{R}_i \ \leftarrow \widetilde{\mathsf{F}}^{\ell_i}_{(i)}(x_{1,..,i},\ \mathsf{layer}{+}1) \in \mathbb{F}_q^{\ell_i \times n}.$$

5. Compute:

$$\mathbf{y} \ \leftarrow \ \mathbf{v} \prod_{i=1}^{k'} \mathbf{M}_{i,x_i} + \sum_{i=1}^{k'} \mathbf{e}_i \cdot \prod_{j=i+1}^{k'} \mathbf{M}_{j,x_j} \in \mathbb{F}_q^n,$$

6. Return $\mathbf{y} + \mathbf{p} \in \mathbb{F}_q^n$.

Figure 9: Hybrid H_1.

4.6 Key-Homomorphism Analysis

We now introduce KH-structure, which only requires that a related property holds with high probability over random keys.

Definition 17 (KH-structure [DJK25])**.** Let $\gamma \in (0, 1/2)$. Let $(\mathcal{K}, \oplus)$ be a group. Let $F \colon \mathcal{K} \times \mathcal{X} \to \mathbb{F}_q$ be a PRF. We say F has γ-KH structure if there exists a tuple (f, h, z) such that for any $k \in \mathcal{K}, x \in \mathcal{X}$, F can be represented as

$$F(k, x) = \langle f(x), h(k) \rangle + z(k, x), \tag{6}$$

where

- $f \colon \mathcal{X} \to \mathbb{F}_q^m$ can be any function.

Hybrid H_2: $\widetilde{\mathsf{G}}$

Global Parameters: Max recursion depth τ, iterators, denoted by $\mathsf{Iter}_{(i)}$ for $i \in [k]$, and

$$\overline{\mathbf{M}}_{(l,r)} \in \mathbb{F}_q^{w_{l-1} \times w_r}, \text{ for } l \in [k],\ r \in [l,k], \qquad \overline{\mathbf{V}}_{(i)},\ \overline{\mathbf{P}}_{(i)} \in \mathbb{F}_q^{T \times w_0} \times \mathbb{F}_q^{T \times w_i}, \text{ for } i \in [k]$$

For each input length $i \in [k]$.

$$R_{(i)} = (R_1, ..., R_T), \text{ where each } R_j \text{ is a random function } R_j \colon [\lambda]^i \to \mathbb{F}_q^n$$

For every $k' = 1, \ldots, k$, define $\widetilde{\mathsf{G}}_{(k')}$ as follows.

Function $\widetilde{\mathsf{G}}_{(k')}(x \in [\lambda]^{k'},\ \mathsf{layer})$:

1. If $\mathsf{layer} = \tau$, return $R(x) \in \mathbb{F}_q^n$, where $R \leftarrow \mathsf{Iter}'_{(k')}.\mathsf{Next}()$.
2. If $\mathsf{layer} = 0$, initialize an iterator $\mathsf{Iter}_{(k')}.\mathsf{Init}(\overline{\mathbf{V}}_{(k')}, \overline{\mathbf{P}}_{(k')}, R_{(k')})$.
3. $(\overline{\mathbf{v}}, \overline{\mathbf{p}}) \leftarrow \mathsf{Iter}_{(k')}.\mathsf{Next}() \in \mathbb{F}_q^{w_0} \times \mathbb{F}_q^{w_{k'}}$.
4. For each input prefix $i \in [k']$:
$$\overline{\mathbf{e}}_i \leftarrow \mathsf{S\text{-}Ber}_{\mu_i}(\overline{\mathbf{R}}_i) \in \mathbb{F}_q^{w_i}, \text{where} \quad \overline{\mathbf{R}}_i \leftarrow \widetilde{\mathsf{G}}_{(i)}^{\ell_i}(x_{1,..,i},\ \mathsf{layer}{+}1) \in \mathbb{F}_q^{\ell_i \times w_i}.$$
5. Compute
$$\overline{\mathbf{y}} \leftarrow \overline{\mathbf{v}} \cdot \overline{\mathbf{M}}_{(1,k')} + \sum_{i=1}^{k'} \overline{\mathbf{e}}_i \cdot \overline{\mathbf{M}}_{(i+1,k')} \in \mathbb{F}_q^{w_{k'}}.$$
6. Return $\overline{\mathbf{y}} + \overline{\mathbf{p}} \in \mathbb{F}_q^{w_{k'}}$.

Figure 10: Hybrid H_2.

- $h \colon \mathcal{K} \to \mathbb{F}_q^m$ is a group homomorphism.
- $z \colon \mathcal{K} \times \mathcal{X} \to \mathbb{F}_q$ is a function satisfying:
  1. **(Efficiency)** For any $x \in \mathcal{X}, k \in \mathcal{K}$, $z(k,x)$ can be computed as efficiently as $F(k,x)$.
  2. **(Vanishing)** For any $x \in \mathcal{X}$, $\Pr_{k \leftarrow \mathcal{K}}[z(k,x) = 1] \le \gamma$.

PRFs with KH property can be transformed into KH-PRFs.

Lemma 6 (KH structure implies KH-PRFs [DJK25]). *Let $n, m \in \mathbb{N}$. If there exists a PRF $F \colon \mathcal{K} \times \mathcal{X} \to \mathbb{F}_q$ with γ-AKH structure, then there exists a 6γ-AKH-PRF $F' \colon \mathcal{K}^n \times \mathcal{X} \to \mathbb{F}_q^n$ such that for any PRF adversary $\mathcal{A}$, it holds that*

$$\mathsf{Adv}_{F'}(\mathcal{A}, n, m) \le m \cdot \mathsf{Adv}_F(\mathcal{A}) + \exp\left(\log|\mathcal{X}| - \frac{m\gamma}{3}\right).$$

Moreover, if F can be computed in depth d, then F' can be computed in depth $2d + O(\log m)$.

The constructed PRFs have KH structure, which then implies KH-PRFs by Lemma 6.

Theorem 10 (KH structure). PRF *has γ-KH structure (Definition 17) where $\gamma = n^{-o(1)}$.*

Proof. We view $\mathbf{M}$ as public parameters and observe that the output of the PRF takes the form $\langle \mathbf{v}, \mathbf{M} \rangle + \mathbf{P} + \sum_{i=1}^{k} \mathbf{e}_i \cdot \mathbf{M}_\star$, which can be written as $\langle \mathbf{v} \,\|\, \mathbf{P}, \mathbf{M} \,\|\, 1 \rangle + \mathbf{e}$, where $\mathbf{e} = \sum \mathbf{e}_i \cdot \mathbf{M}_\star$. It then follows that $\mathbf{e}$ is zero except with probability $n^{-o(1)}$ over the randomness of secret key.

□

4.7 Putting Everything Together

We are now ready to prove our meta theorem.

Proof of Theorem 6. If $\mathsf{iMatMul}$ and Reach can be deterministically computed in $O(\log \lambda)$ depth, then combining the above theorems immediately leads to Theorem 6.

It remains to consider the case where $\mathsf{iMatMul}$ and Reach computed by randomized $O(\log \lambda)$-depth circuits. By sufficiently amplifying the correctness via majority voting (which can be done in $O(\log \lambda)$ depth), for every $\mathbf{k}, x$, $\mathsf{Eval}_{\mathbf{k}}(x; r) = \mathsf{PRF}_{\mathbf{k}}(x)$ except with probability $2^{-\mathsf{poly}(\lambda)}$ over the randomness of Eval. Then one can define a final PRF' whose key additionally includes the random bits required by Eval. Then it holds that for a random key $(\mathbf{k}, r)$[17],

$$\mathsf{PRF}'_{\mathbf{k},r}(\cdot) \coloneqq \mathsf{Eval}_{\mathbf{k}}(\cdot; r) \approx_{stat.} \mathsf{PRF}_{\mathbf{k}}(\cdot) \approx_{compt.} R(\cdot)$$

the first $\approx_{stat.}$ is by taking a union bound over all input, so for a random r, $\mathsf{Eval}_{\mathbf{k}}(x; r) = \mathsf{PRF}_{\mathbf{k}}(x)$ for every x except with probability $2^{-\mathsf{poly}(\lambda)} \cdot |\mathcal{X}| = \mathsf{negl}(\lambda)$. □

5 Instantiation I: NC^1 PRFs from Polynomial Hardness of $\mathsf{SparseLPN}$

5.1 The $\mathsf{SparseLPN}$ Assumption

In this section, we first recall expanders, then recall the $\mathsf{SparseLPN}$ assumption [Ale03, Ale11].

Unbalanced expanders. We introduce the notion of *unbalanced (bipartite) expanders*. From now on, we will simply refer to them as expanders.

Definition 18 (Unbalanced Expander)**.** Let $n \in \mathbb{N}$, $c_1 > 1$, $c_2 \in (0, 1)$,

$$m = n^{1.01}, \quad d = \log^{c_1} n, \quad m' = n^{c_2},$$

Define a neighbor function[18]

$$\Gamma \colon [m] \times [d] \to [n],$$

[17] As part of the PRF key, once sampled, the same r is reused for different PRF inputs x.

[18] Note that the input length to this function is $O(\log n)$.

which induces a left-regular bipartite graph of left-degree d. For a vertex $u \in [m]$, let

$$\Gamma(u) \coloneqq \{\Gamma(u,i) : i \in [d]\} \subseteq [n],$$

and for a set $S \subseteq [m]$ define $\Gamma(S) \coloneqq \bigcup_{u \in S} \Gamma(u)$.

We say that Γ is a (n, m, d, m')-expander if for every $S \subseteq [m]$ with $|S| \leq m'$,

$$|\Gamma(S)| \ > \ \frac{1}{2} \cdot d\,|S|.^{19}$$

The bipartite expander graph induced by Γ is $G_\Gamma = (L \cup R, E)$ with

$$E \ = \ \{(u, \Gamma(u,i)) : u \in [m], \ i \in [d]\}.$$

The expander matrix $\mathbf{A}_\Gamma \in \{0,1\}^{n \times m}$ induced by Γ is defined by

$$(\mathbf{A}_\Gamma)_{v,u} \ = \ 1 \quad \iff \quad v \in \Gamma(u).$$

We will use $\Gamma_\$$ to denote random expanders, i.e., a uniformly random map $[L] \times [d] \to [R]$. Otherwise, Γ is assumed to be an explicit expander.

Definition 19 (SparseLPN [Ale03, Ale11])**.** Let $n, m, d, t \in \mathbb{N}, d < n$, $\mu \in (0, 1/2), \epsilon \in (0,1)$. Let $\Gamma \colon [m] \times [d] \to [n]$ be the (n, m, d, m')-expander. The $[n, m, \mu, t, \epsilon]_2$-$\mathsf{SparseLPN}_\Gamma$ assumption states that

$$(\mathbf{A}_\Gamma, \mathbf{s} \cdot \mathbf{A}_\Gamma + \mathbf{e}) \approx_{t,\epsilon} (\mathbf{A}_\Gamma, \mathbf{u}), \tag{7}$$

where $\mathbf{s} \leftarrow\!\$\ \mathbb{F}_2^n$, $\mathbf{A}_\Gamma \in \mathbb{F}_2^{n \times m}$ is the expander matrix induced by Γ, $\mathbf{e} \leftarrow\!\$\ \mathsf{Ber}_\mu^m$, and $\mathbf{u} \leftarrow\!\$\ \mathbb{F}_2^m$.

Assumption 1 (Polynomial hardness of SparseLPN (with respect to a sublinear-depth expander))**.** Let $m = m(n), \mu = \mu(n), d = d(n), t = t(n), \epsilon = \epsilon(n)$ all be functions of $n \in \mathbb{N}$. We say SparseLPN is polynomially hard if there exists $m(n) = n^c$ for $c > 1$, $d(n) = \mathsf{poly}(\log n)$, and a sublinear-depth expander $\Gamma : [m] \to [n]^d$ (namely, computable in depth $o(\log n)$), such that for any noise rate

$$\mu(n) = \Omega(n^{-1/\log\log n}),^{20}$$

and any $t(n) = \mathsf{poly}(n)$, there exists a negligible $\epsilon(n) = \mathsf{negl}(n)$, such that $[n, m, \mu, t, \epsilon]_2$-$\mathsf{SparseLPN}_\Gamma$ assumption holds for all sufficiently large n.

From now on, when we say SparseLPN, we actually mean SparseLPN with respect to a sublinear-depth expander.

Remark 7. *(Uniformity) Our construction will use a non-uniform advice: a family of sublinear-depth expanders satisfying Assumption 1. We can eliminate the use of the advice by using an explicit variant of Assumption 1. In particular, we can assume that Assumption 1 holds for a specific construction of sublinear-depth expanders [GUV09, KT22].*

Remark 8 (On well-foundedness of our SparseLPN assumption)**.** The original SparseLPN assumption [Ale03] posits that it holds with respect to any expander with sufficiently good expansion (assumption

[19] This is an expansion regime where a certain property called "unique neighbors" is guaranteed, which will then guarantee that the minimal distance of the expander codes induced by Γ will be $m' + 1$.

[20] We note that this quantity is only slightly sub-constant, and larger than any inverse polynomial

($\star$)). As mentioned, assumption ($\star$) is unconditionally secure against linear tests. This is an active area of research. Recent works have provided cautionary evidence that ($\star$) *might* be false [OST19, OST22, BLRS26], by giving attacks for specific choices of expanders. These attacks either require quasi-polynomially many samples or rely on additional assumptions for their algorithms to work. Our results rely only on the weaker assumption that there exists a sublinear-depth expander for which the resulting SparseLPN assumption holds (assumption ($\star\star$)). To better contextualize this assumption, let us compare it with DDH. The DDH assumption is certainly false if stated for arbitrary groups, since there are many easy groups in which DDH does not hold. By contrast, for SparseLPN, it remains open whether there is an unconditional counterexample to assumption ($\star$). Therefore, we view our assumption ($\star\star$) as conservative.

We conclude this section with a fact that SparseLPN unconditionally holds against a family of (potentially unbounded) distinguishers called "linear test," which capture almost all existing attacks against (variants of) LPN assumptions. This follows because the dual distance of the underlying codes, namely, the duals of the expander codes induced by Γ, equals the minimum distance of the expander codes themselves, which is $m' = n^c$ for some constant c. Then the distinguishing advantage of linear test is $2^{-\Omega(n^{c'})}$ for some constant c'.

5.2 Walking on Expander Graphs

$\mathsf{MultReach}(u, \mathbf{M}_{x_1}, \ldots, \mathbf{M}_{x_k})$

Parameter: Expander matrix $\mathbf{M}_\Gamma = (\mathbf{M}_1, \ldots, \mathbf{M}_\lambda) \in \{0,1\}^{n \times n\lambda}$ induced by $\Gamma \colon [n \cdot \lambda] \to [n]^d$.

On input $(u, \mathbf{M}_{x_1}, \ldots, \mathbf{M}_{x_k})$, where $u \in [n], x_{k'} \in [\lambda]$ for $k' \in [k]$:

1. Initialize the list $\mathbf{c}^{(0)} \leftarrow (u) \in [n]$.
2. For $i = 1, ..., k$ sequentially:
   (a) Replace each entry $u \in [n]$ of $\mathbf{c}^{(i-1)}$ by its d neighbors at the next layer

$$\mathbf{c}^{(i)} \leftarrow \left(\phi_i(u) \ : \ u \in \mathbf{c}^{(i-1)}, \right) \in [n]^{d^i},$$

   where each

$$\phi_i(u) \coloneqq \Gamma\left(u_{(i)}\right) \quad f_\Gamma(\log n\lambda) \text{ depth}$$
$$\text{where } u_{(i)} \leftarrow u + n \cdot (x_i - 1) \in [n\lambda] \quad O(\log\log(n\lambda)) \text{ depth}$$

3. Output $(\mathbf{c}^{(0)}, \ldots, \mathbf{c}^{(k)})$

Depth:

$$O\left(k \cdot (o(\log n\lambda) + \log\log(n\lambda))\right) = O(\log \lambda).$$

Figure 11: Reachability (with Multiplicity) over $(k+1)$-Layered Graph.

$\mathsf{Reach}(u, \mathbf{M}_{x_1}, \ldots, \mathbf{M}_{x_k})$

Parameter: Expander matrix $\mathbf{M}_\Gamma = (\mathbf{M}_1, \ldots, \mathbf{M}_\lambda) \in \{0,1\}^{n \times n\lambda}$ induced by $\Gamma \colon [n \cdot \lambda] \to [n]^d$.

On input $(u, \mathbf{M}_{x_1}, \ldots, \mathbf{M}_{x_k})$, where $u \in [n], x_{k'} \in [\lambda]$ for $k' \in [k]$:

1. $(\mathbf{c}^{(0)}, \ldots, \mathbf{c}^{(k)}) \leftarrow \mathsf{MultReach}(u, x_1, ..., x_k)$ (Figure 11).
2. For $i \in \{0, ..., k\}$ in parallel:
   (a) Initialize $\mathbf{a} \leftarrow \mathbf{0} \in \mathbb{F}_2^n$.
   (b) Check whether each node v is reachable or not for $v \in [n]$:

$$\mathbf{a}_v \leftarrow \bigvee_{u \in \mathbf{c}^{(i)}} \mathbb{I}[u = v] \text{ (indicator function)}, \quad O\left(\log \prod_{i=1}^{k} d_i\right) \text{ depth}$$

   (c) $\mathbf{c}^{(i)} \leftarrow \mathsf{Supp}(\mathbf{a};\ w_i) \in [n]^{w_i}$ (Definition 7) $\quad O(\log n)$ depth
3. Output $(\mathbf{c}^{(0)}, \ldots, \mathbf{c}^{(k)})$

Depth:

$$O\left(k \cdot (f_\Gamma(\log n\lambda) + \log\log(n\lambda)) + \sum_{i=1}^{k} \log d_i + \log n\right)$$

Figure 12: Reachability over Layered Expander Graph.

Let $\Gamma \colon [\lambda n] \to [n]^d$ be an expander. Denote the depth of Γ by $f_\Gamma(\log \lambda n)$. Γ induces an adjacency matrix $\mathbf{M}_\Gamma = (\mathbf{M}_1, ..., \mathbf{M}_\lambda) \in \mathbb{F}_2^{n \times n\lambda}$. Observe that the support of j-th column $\mathbf{M}_i$ is given by $\Gamma(j + (i-1) \cdot n) \in [n]^d$. In other words, the neighbor function induced by $\mathbf{M}_i$ is $\phi_{\mathbf{M}_i}(j) = \Gamma(j + (i-1) \cdot n)$.

We show how to compute Reach (Definition 13) and $\mathsf{iMatMul}$ (Definition 12) over $\mathcal{M} = \mathbf{M}_\Gamma$.

To enable a modular treatment, we first present $\mathsf{MultReach}$ in Figure 11, which returns a multiset (unordered and possibly with duplicates) of reachable nodes. Procedure (2) can be viewed as a breadth-first search (BFS) starting at u for k steps. The list $\mathbf{c}^{(i)}$ collects all nodes reachable by walks of exactly i steps[21].

Reachability. We then define Reach (Figure 12), which first computes $\mathsf{MultReach}$, then sorts, deduplicates, and pads $\mathbf{c}^{(i)}$ to length w using a bucket/bin-style routine. Combining both phases, the overall circuit depth of Reach is

$$O\left(k \cdot (f_\Gamma(\log n\lambda) + \log\log(n\lambda)) + \sum_{i=1}^{k} \log d_i + \log n\right)$$

[21] Since it is a layered graph, all nodes reachable by t steps are at layer t

$\mathsf{iMatMul}(\mathbf{M}_{x_1}, \ldots, \mathbf{M}_{x_k})$

Parameter: Expander matrix $\mathbf{M}_\Gamma = (\mathbf{M}_1, \ldots, \mathbf{M}_\lambda) \in \{0,1\}^{n \times n\lambda}$ induced by $\Gamma \colon [n \cdot \lambda] \to [n]^d$.

On input $(\mathbf{M}_{x_1}, \ldots, \mathbf{M}_{x_k})$, where $x_{k'} \in [\lambda]$ for $k' \in [k]$:

1. For each $j \in [n]$ in parallel:
   (a) $(\mathbf{c}^{(0)}, \ldots, \mathbf{c}^{(k)}) \leftarrow \mathsf{MultReach}(j, x_k, ..., x_1)$ (Figure 11).
   (b) Initialize $\mathbf{A} \leftarrow \mathbf{0} \in \mathbb{F}_2^{n \times n}$.
   (c) For each $i \in [n]$ in parallel:
      i. Counting (mod 2) the number of paths from i to j:

$$\mathbf{A}[i,j] \leftarrow \sum_{u \in \mathbf{c}^{(k)}} \mathbb{I}[u = i] \text{ (indicator function)}, \quad O\left(\log \prod_{i=1}^{k} d_i\right) \text{ depth}$$

2. Output $\mathbf{A} \in \mathbb{F}_2^{n \times n}$.

Depth:

$$O\left(k \cdot (f_\Gamma(\log n\lambda) + \log\log(n\lambda)) + \sum_{i=1}^{k} \log d_i\right)$$

Figure 13: Iterated Matrix Multiplication.

Iterated matrix multiplication via path counting. We will rely on the observation that iterated matrix multiplication over $\mathbb{F}_2$ is equivalent to counting the number of paths (modulo 2) between each pair of nodes, which then reduces to reachability with multiplicity (i.e., $\mathsf{MultReach}$ we constructed before), and then use a bin-style approach to count (modulo 2) the occurrence of every node $i \in [n]$ in the reachability set. The overall circuit depth of $\mathsf{iMatMul}$ is

$$O\left(k \cdot (f_\Gamma(\log n\lambda) + \log\log(n\lambda)) + \sum_{i=1}^{k} \log d_i\right)$$

5.3 Configuration

Configuration 1 (NC^1 PRFs from $\mathsf{SparseLPN}$)**.** Let λ be a security parameter. Let $\Gamma \colon [n^c] \to [n]^d$, where $c > 1$, $d = \mathsf{poly}(\log n)$, be an expander family for which Assumption 1 holds. Denote the depth of Γ by $f_\Gamma(\log n) = o(\log n)$.

1. Dimension $n = \lambda^{1/(c-1)}$, modulus $q = 2$.
2. Maximum recursion depth $\tau = (\log \lambda)^{1/2}$.
3. Number of input chunks $k = \min(\log\log\log \lambda, \log n / f_\Gamma(\log n))$.

(a) For $i \in [k]$, noise rate $\mu_i = \Theta(n^{-1/\log\log n})$ (chosen to be an inverse power of two). Let $\ell_i = \log(1/\mu_i)$. Let Bernoulli sample function $\mathsf{S\text{-}Ber}_{\mu_i} \colon \mathbb{F}_2^{\ell_i} \to \mathbb{F}_2$:

$$\mathsf{S\text{-}Ber}_{\mu_i}(\mathbf{x}) := \prod_{j=1}^{\ell_i} x_j.$$

(b) For $i \in [k]$, sparsity $d_i = d$, and distribution $\mathcal{M}_i = \mathbf{A}_\Gamma \in \{0,1\}^{n \times n\lambda}$.

Corollary 1 (NC^1 **PRF from Polynomial Hardness of** $\mathsf{SparseLPN}$)**.** *Let λ be a security parameter. Assume the polynomial hardness of* $\mathsf{SparseLPN}$ *with respect to an expander family computable in sub-linear depth (Assumption 1). There exists a secure KH-PRF family.*

Proof. It suffices to show that Configuration 1 is a valid key-homomorphic configuration (Definition 14), then the corollary follows from Theorem 6.

1. $k = \min(\log\log\log\lambda, \log n / f_\Gamma(\log n)) = \omega(1)$, as desired.
2. $\log k + \log\log q = O(\log k) = O((\log\lambda)/\tau)$, as desired.
3. For $i \in [k]$, $\mathsf{S\text{-}Ber}_{\mu_i}$ can be computed in depth
$$O(\log \ell_i) = O(\log\log(1/\mu_i)) = O(\log\log n) = O((\log\lambda)/\tau),$$
as desired.
4. It holds that
$$\min(k, \tau) \cdot \sum_{i=1}^{k} \log d_i = O(\log\log\log^2\lambda \cdot \log\log n) = O(\log\lambda),$$
as desired.
5. We need to show
$$\begin{aligned} &-\tau \cdot \log\left(\sum_{i=1}^{k} 2\mu_i (k\ell)^\tau \cdot \prod_{j=i+1}^{k} d_j\right) = \omega(k\log\lambda) \\ \Longleftarrow\ &\tau \cdot (\log(1/\mu) - \log k - \tau\log(k\ell) - k\log d) = \omega(\log\log\log n \cdot \log n) \\ \Longleftarrow\ &(\log n)^{1/2} \cdot \left(\frac{\log n}{\log\log n} - o\left(\frac{\log n}{\log\log n}\right)\right) = \omega(\log\log\log n \cdot \log n), \end{aligned}$$
as desired.
6. $\mathsf{iMatMul}$ can be computed in depth
$$O\left(k \cdot (f_\Gamma(\log n\lambda) + \log\log(n\lambda)) + \sum_{i=1}^{k} \log d_i\right) = O(\log\lambda)$$
7. Reach can be computed in depth
$$O\left(k \cdot (f_\Gamma(\log n\lambda) + \log\log(n\lambda)) + \sum_{i=1}^{k} \log d_i + \log n\right) = O(\log\lambda)$$

8. It follows from Assumption 1 (via a standard hybrid argument for multiple independent secrets) that for every $i \in [k]$: For any polynomial $m = \mathsf{poly}(\lambda)$,

$$(\mathbf{M}_i, \mathbf{V} \cdot \mathbf{M}_i + \mathbf{E}) = (\mathbf{A}_\Gamma, \mathbf{V} \cdot \mathbf{A}_\Gamma + \mathbf{E}) \approx_{compt.} (\mathbf{A}_\Gamma, \mathbf{U}) = (\mathbf{M}_i, \mathbf{U}),$$

where $\mathbf{V} \leftarrow\!\$\, \mathbb{F}_q^{m \times n}, \mathbf{M}_i \leftarrow\!\$\, \mathcal{M}_i \in \mathbb{F}_q^{n \times \lambda n}$, $\mathbf{E} \leftarrow\!\$\, \mathsf{Ber}_{\mu_i}^{m \times \lambda \cdot n}, \mathbf{U} \leftarrow\!\$\, \mathbb{F}_q^{m \times \lambda \cdot n}$.

□

6 Instantiation II: NC^1 PRFs from Polynomial Hardness of RingLPN

6.1 The RingLPN Assumption

In this section, we recall the RingLPN assumption, which was first introduced (over $\mathbb{F}_2$) in [HKL+12]. Since then, it has received growing attention from the cryptography community [BL12, DP12, LP15, GJL15, BCG+20, LWYY24], due to its appealing efficiency advantages over LPN. We will be mainly interested in RingLPN over cyclotomic rings, introduced in [BCG+20], which have previously been used for Ring-LWE [LPR10, LPR13a]. For discussions regarding existing attacks on RingLPN, we refer the reader to [BCG+20].

Definition 20 (RingLPN [HKL+12, BCG+20])**.** Let $n, m, t \in \mathbb{N}$, n be a power of two, $\mu \in (0, 1/2), \epsilon \in (0, 1)$. Let $\mathcal{R}$ be cyclotomic polynomial ring $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$. Let q be a prime such that $q = 1 \mod 2n$. Define the quotient ring $R_q = R/qR$. Let χ_μ be the noise distribution over $\mathcal{R}_q$ whose coefficient is distributed as $\mathsf{Ber}_\mu(\mathbb{F}_q)$. The $[n, m, \mu, t, \epsilon]_{\mathcal{R}_q}$-RingLPN assumption states that

$$(\mathbf{a}, s \cdot \mathbf{a} + \mathbf{e}) \approx_{t,\epsilon} (\mathbf{a}, \mathbf{u}),$$

where $s \leftarrow\!\$\, \mathcal{R}_q$, $\mathbf{a} \leftarrow\!\$\, \mathcal{R}_q^m$, $\mathbf{e} \leftarrow\!\$\, \chi_\mu^m$, and $\mathbf{u} \leftarrow\!\$\, \mathcal{R}_q^m$. Or equivalently,

$$(\mathbf{A}, \mathbf{s} \cdot \mathbf{A} + \mathbf{e}) \approx_{t,\epsilon} (\mathbf{A}, \mathbf{u}),$$

where $\mathbf{A} \leftarrow \mathsf{Mat}(a_1) \,||\, \ldots \,||\, \mathsf{Mat}(a_m) \in \mathbb{F}_q^{n \times nm}$ (Definition 21) for $a_i \leftarrow\!\$\, \mathcal{R}_q$, $\mathbf{s} \leftarrow\!\$\, \mathbb{F}_q^n$, $\mathbf{e} \leftarrow\!\$\, \mathsf{Ber}_\mu^{nm}(\mathbb{F}_q)$, $\mathbf{u} \leftarrow\!\$\, \mathbb{F}_q^{nm}$,

Definition 21 (Vec and Mat)**.** Let all parameters be as in Definition 20. Write $a \in \mathcal{R}_q$ in the monomial basis as $a = \sum_{i=0}^{n-1} a_i X^i$, $a_i \in \mathbb{F}_q$. Define the coefficient vector embedding

$$\mathsf{Vec}\colon \mathcal{R}_q \to \mathbb{F}_q^n, \quad \mathsf{Vec}(a) \coloneqq (a_0, \ldots, a_{n-1}).$$

For $b \in \mathcal{R}_q$, define the coefficient matrix embedding

$$\mathsf{Mat}\colon \mathcal{R}_q \to \mathbb{F}_q^{n \times n}, \quad (\mathsf{Mat}(b))_{i,*} \coloneqq \mathsf{Vec}(X^{i-1} \cdot b), \text{ for } i \in [n].$$

Then for all $a, b \in \mathcal{R}_q$ it holds that

$$\mathsf{Vec}(a \cdot b) = \mathsf{Vec}(a) \cdot \mathsf{Mat}(b).$$

Lemma 7. *For all $a_1, a_2 \in \mathcal{R}_q$, it holds that*

$$\mathsf{Mat}(a_1 \cdot a_2) = \mathsf{Mat}(a_1) \cdot \mathsf{Mat}(a_2).$$

Proof. By the defining property of Mat, for every $x \in \mathcal{R}_q$,

$$\mathsf{Vec}(x \cdot b) = \mathsf{Vec}(x) \cdot \mathsf{Mat}(b).$$

Applying this identity twice, we obtain

$$\mathsf{Vec}(x) \cdot \mathsf{Mat}(a_1) \cdot \mathsf{Mat}(a_2) = \mathsf{Vec}(x \cdot a_1) \cdot \mathsf{Mat}(a_2) = \mathsf{Vec}((x \cdot a_1) \cdot a_2).$$

By associativity of multiplication in $\mathcal{R}_q$, this is

$$\mathsf{Vec}(x \cdot (a_1 \cdot a_2)).$$

Using the defining property again, we also have

$$\mathsf{Vec}(x \cdot (a_1 \cdot a_2)) = \mathsf{Vec}(x) \cdot \mathsf{Mat}(a_1 \cdot a_2).$$

Therefore, for every $x \in \mathcal{R}_q$,

$$\mathsf{Vec}(x) \cdot \mathsf{Mat}(a_1) \cdot \mathsf{Mat}(a_2) = \mathsf{Vec}(x) \cdot \mathsf{Mat}(a_1 \cdot a_2).$$

Taking $x = 1, X, \ldots, X^{n-1}$, whose coefficient vectors form the standard basis of $\mathbb{F}_q^n$, implies

$$\mathsf{Mat}(a_1) \cdot \mathsf{Mat}(a_2) = \mathsf{Mat}(a_1 \cdot a_2).$$

□

An equivalent view of $\mathsf{RingLPN}$ is that it postulates $(\mathbf{A}, \mathbf{s} \cdot \mathbf{A} + \mathbf{e})$ is pseudorandom, where $\mathbf{A} = (\mathbf{A}_1, \ldots, \mathbf{A}_m)$ with each block $\mathbf{A}_i$ generated as the following procedure: its first row is generated uniformly at random in $\mathbb{F}_q^n$, then the remaining rows of $\mathbf{A}_i$ are obtained by a cyclic shift of the previous row with a sign flip on the wrapped-around entry (i.e., a negacyclic shift). Since each block of $\mathbf{A}$ is negacyclic, $\mathsf{RingLPN}$ can be viewed as the problem of decoding random quasi-negacyclic codes.

We now formally define the polynomial hardness of $\mathsf{RingLPN}$ (or $\mathsf{RingLPN}^{s\sim\chi}$), which says that for noise rate of $\mu = \Omega(\frac{\log^c n}{n})$, $\mathsf{RingLPN}$ (or $\mathsf{RingLPN}^{s\sim\chi}$) holds against any polynomial time distinguisher.

Assumption 2 (Polynomial Hardness of $\mathsf{RingLPN}$ and $\mathsf{RingLPN}^{s\sim\chi}$)**.** Let $m = m(n), \mu = \mu(n), q = q(n), t = t(n), \epsilon = \epsilon(n)$ all be functions of $n \in \mathbb{N}$. Let c be an arbitrarily large constant. We say (low-noise) $\mathsf{RingLPN}$ (or $\mathsf{RingLPN}^{s\sim\chi}$) is polynomially hard (or hard, for short) if for any noise rate

$$\mu(n) = \Omega\left(\frac{\log^c n}{n}\right),$$

and any

$$m(n) = \mathsf{poly}(n), \quad t(n) = \mathsf{poly}(n),$$

there exists a negligible $\epsilon(n) = \mathsf{negl}(n)$, such that the $[n, m, \mu, t, \epsilon]_{\mathcal{R}_q}$-$\mathsf{RingLPN}$ (or $[n, m, \mu, t, \epsilon]_{\mathcal{R}_q}$-$\mathsf{RingLPN}^{s\sim\chi}$) assumption holds for all sufficiently large n.

Different to the previous noise rate $\mu = \Omega(n^{-1/\log\log n})$ used in LPN, for $\mathsf{RingLPN}$ we will use an extremely low-noise rate $\mu = \Omega(\frac{\log^c n}{n})$, but only needs to assume the resulting $\mathsf{RingLPN}$

is polynomially hard. In this noise rate regime, the assumption is still believed to hold for any polynomial $m = \mathsf{poly}(n)$. However, it can only achieve security against adversaries that run in quasi-polynomial time $2^{o(\log^2 n)}$ due to Gaussian elimination [EKM17]. This low noise rate was used in several previous works [BLVW19, AMR25].

6.2 Reduction to RingLPN with Sparse Secrets

We now introduce $\mathsf{RingLPN}^{s\sim\chi}$, where the secret is chosen from the noise distribution.

Definition 22 ($\mathsf{RingLPN}^{s\sim\chi}$)**.** Let all parameters be as in Definition 20. The $[n, m, \mu, t, \epsilon]_{\mathcal{R}_q}$-$\mathsf{RingLPN}^{s\sim\chi}$ assumption states that

$$(\mathbf{a}, \mathbf{a} \cdot s + \mathbf{e}) \approx_{t,\epsilon} (\mathbf{a}, \mathbf{u}), \tag{8}$$

where $s \leftarrow\!\$\, \chi_\mu$, $\mathbf{a} \leftarrow\!\$\, \mathcal{R}_q^m$, $\mathbf{e} \leftarrow\!\$\, \chi_\mu^m$, and $\mathbf{u} \leftarrow\!\$\, \mathcal{R}_q^m$. Or equivalently,

$$(\mathbf{A}, \mathbf{A} \cdot \mathbf{S} + \mathbf{E}) \approx_{t,\epsilon} (\mathbf{A}, \mathbf{U}),$$

where $\mathbf{A} \leftarrow\!\$\, \mathbb{F}_q^{m\times n}$, $\mathbf{S} = \mathsf{Mat}(s) \in \mathbb{F}_q^{n\times n}$ for $s \leftarrow\!\$\, \chi_\mu$, $\mathbf{E} \leftarrow\!\$\, \mathsf{Ber}_\mu^{m\times n}$, $\mathbf{U} \leftarrow\!\$\, \mathbb{F}_q^{m\times n}$.

We will rely on a reduction from RingLPN to $\mathsf{RingLPN}^{s\sim\chi}$, which is essentially identical to its RingLWE analogue in [LPR13b].

Lemma 8. *If* RingLPN *is polynomially hard, then* $\mathsf{RingLPN}^{s\sim\chi}$ *is also polynomially hard.*

Proof. See Appendix A.4. □

Lemma 9. *Let $\mu \in (0, 1)$. Let $s \leftarrow\!\$\, \chi_\mu$. Then each column of $\mathsf{Mat}(s)$ is $d \le 2n\mu$ sparse except with probability $2^{-\Omega(n\mu)}$.*

Proof. Write

$$s = \sum_{j\in S} f_j X^{j-1},$$

where $S \subseteq [n]$. By a standard Chernoff bound, $|S| = d \le 2n\mu$ except with probability $2^{-\Omega(n\mu)}$. By definition, the i-th row of $\mathsf{Mat}(s)$ is the coefficient representation of

$$X^{i-1} \cdot s = \sum_{j\in S} f_j X^{i+j-2} = \sum_{j\in S} f'_j X^{(i+j-2) \bmod n} \pmod{X^n + 1}.$$

Hence each row has at most $|S| \le d$ nonzero entries.

It remains to prove column sparsity. Fix a column index $j \in [n]$. The $\mathsf{Mat}(s)$ can be nonzero only if there exists some $k \in [n]$ such that

$$k \equiv i + j - 2 \pmod{n}.$$

Equivalently,

$$i \equiv k - j + 2 \pmod{n}.$$

For every fixed $j \in [n]$, there is exactly one row index $i \in [n]$ satisfying this congruence. It follows that column j has at most $|S| \le d$ nonzero entries. □

6.3 Bernoulli Sample Function over $\mathbb{F}_q$

In this section, we introduce a central gadget in our constructions, the Bernoulli sample function $\mathsf{S\text{-}Ber}$. We first give its definition and then prove its key properties.

Definition 23 ((Approximate) Bernoulli Sample Function over $\mathbb{F}_q$)**.** Let $\mu \in (0,1)$ be an inverse power of 2. Let $\ell = \log(1/\mu)$. Let $q \in \mathbb{Z}$ such that $\log q > \ell$. For an input $\mathbf{x} = (x_1, x_2) \in \mathbb{F}_q^2$, the approximate Bernoulli sample function $\mathsf{S\text{-}Ber}'_\mu : \mathbb{F}_q^2 \to \mathbb{F}_q$ is defined as follows.

$$\mathsf{S\text{-}Ber}'_\mu(\mathbf{x}) = \mathbb{I}[(x_1)_{1,\ldots,\ell} = \mathbf{0}^\ell] \cdot x_2, \tag{9}$$

where $(x_1)_{1,\ldots,\ell} \in \{0,1\}^\ell$ is the least ℓ significant bits of x_1.

Lemma 10. *It holds that*

$$\mathsf{SD}(\mathsf{S\text{-}Ber}'_\mu(\mathbf{u}), e) \le 2/q, \tag{10}$$

where $\mathbf{u} \leftarrow \mathbb{F}_q^2$, $e \leftarrow \mathsf{Ber}_\mu(\mathbb{F}_q)$.

Proof. See Appendix A.5. □

Lemma 11. *Let $\mu \in (0,1)$ be an inverse power of 2. $\mathsf{S\text{-}Ber}'_\mu$ is an (exact) Bernoulli sample function for $\mathsf{S\text{-}Ber}_{\mu'}$ with*

$$\mu' = \Pr_{\mathbf{x} \leftarrow\$ \mathbb{F}_q^2}[\mathsf{S\text{-}Ber}'_\mu(\mathbf{x}) \neq 0] = \mu + O(2/q)$$

Proof. The first equality follows from Definition 10. The second equality follows from Lemma 10. □

6.4 Configuration

Configuration 2 (NC^1 PRFs from $\mathsf{RingLPN}$)**.** Let λ be a security parameter.

1. Dimension $n = \lambda$. Let modulus $q = \lambda^{\log\log\lambda}$ be a prime such that $q = 1 \mod 2n$.
2. Maximum recursion depth $\tau = (\log\lambda)^{1/2}$.
3. Number of input chunks $k = \log\log\log\lambda$.
   (a) For $i \in [k]$, let $\mu'_i = \Theta(\frac{\log^{10} n}{n})$ (chosen to be an inverse power of two). Let the noise rate
   $$\mu_i = \Pr_{\mathbf{x} \leftarrow\$ \mathbb{F}_q^2}[\mathsf{S\text{-}Ber}'_{\mu'}(\mathbf{x}) \neq 0].$$
   Let the Bernoulli sample function $\mathsf{S\text{-}Ber}_{\mu_i} := \mathsf{S\text{-}Ber}'_{\mu'} : \mathbb{F}_q^2 \to \mathbb{F}_q$.
   (b) For $i \in [k]$, the distribution $\mathcal{M}_i \in \{0,1\}^{n \times n\lambda}$ is defined as follows: $\mathbf{M} = (\mathbf{M}_1, \ldots, \mathbf{M}_\lambda)$, where $\mathbf{M}_i = \mathsf{Mat}(s_i)$ for $s_i \leftarrow\$ \chi$. The sparsity $d_i = 2n\mu$ (except with negligible probability) by Lemma 9.

Reachability $\mathsf{Reach}(u, m_1, ..., m_k)$

Input: $u \in [n]$, $m_1, ..., m_k \in \mathcal{R}_q$.

1. $\mathbf{c}_{(0)} \leftarrow (u)$.
2. For $i \in [k]$ in parallel:
   (a) $m \leftarrow \prod_{j=1}^{i} m_j \in \mathcal{R}_q$ $O(\log n)$ depth
   (b) $\mathbf{A} \leftarrow \mathsf{Mat}(m) \in \mathbb{F}_q^{n \times n}$ (Definition 21)
   (c) $\mathbf{c}_{(i)} \leftarrow \mathsf{RowSupp}(\mathsf{ColSelect}(\mathbf{A}, u);\ w_i) \in [n]^{w_i}$ $O(\log n)$ depth.
3. Output $(\mathbf{c}_{(0)}, \ldots, \mathbf{c}_{(k)})$.

Depth: $O(\log n)$

Figure 14: Reachability via Iterated Multiplication

Iterated Matrix Multiplication. By Lemma 7, For $\mathbf{M}_1, \ldots, \mathbf{M}_k$ where $\mathbf{M}_k = \mathsf{Mat}(m_k)$, computing $\prod_i^k \mathbf{M}_i$ reduces to computing

$$\mathsf{Mat}\left(\prod_{i=1}^{k} m_i\right).$$

To compute the iterated ring product, we first map the ring elements $m_1, \ldots, m_k$ to their Chinese-remainder representation modulo q (i.e., evaluate the defining polynomials at the n roots of $X^n + 1$ modulo q). In this representation, iterated multiplication becomes coordinate-wise multiplication of k vectors, which is in $O(\log n)$ depth since iterated integer multiplication is in NC^1 [BCH86, HAB02]. Finally, we convert the result back to coefficient representation. The conversion between coefficient representation to evaluation representation is simply a matrix-vector multiplication over $\mathbb{Z}_q$, which is also in NC^1 since both integer multiplication and iterated integer addition are in NC^1. Hence the overall depth is $O(\log n)$.

Reachability Set. Reach is formally specified in Figure 14 and relies on the equivalence between matrix multiplication over $\mathbb{F}_q$ for $q = n^{\omega(1)}$ and reachability in layered graphs. More concretely, the support of the relevant row/column of the product matrix records the vertices reachable from the starting vertex through the corresponding sequence of layers. If the matrix product were computed over the integers with nonnegative entries, then a nonzero entry would exactly indicate the existence of at least one path. In our setting, the computation is performed over $\mathbb{F}_q$. However, since we take $q = n^{\omega(1)}$, the probability of such collisions is $\mathsf{negl}(n)$ when inputs are uniformly random on their support, as in our $\mathsf{RingLPN}$ setting. Taking a union bound over all entries makes the total failure probability negligible.

Corollary 2 (NC^1 **PRFs from polynomial hardness of** $\mathsf{RingLPN}$)**.** *Let λ be a security parameter. Assuming the polynomial hardness of* $\mathsf{RingLPN}$ *(Assumption 2), there exists a secure pseudorandom function family in* NC^1.

Proof. It suffices to show that Configuration 2 is a valid configuration (Definition 14), then the

corollary follows from Theorem 6.

1. $k = \log\log\log\lambda = \omega(1)$, as desired.
2. $\log k + \log\log q = O((\log\lambda)/\tau)$, as desired.
3. For $i \in [k]$, $\mathsf{S\text{-}Ber}_{\mu_i} = \mathsf{S\text{-}Ber}'_{\mu'}$ can be computed in depth
$$O(\log\log q) = O((\log\lambda)/\tau),$$
as desired.
4. It holds that
$$\min(k,\tau)\cdot\sum_{i=1}^{k}\log d_i = O(\log\log\log^2\lambda\cdot\log\log n) = O(\log\lambda),$$
as desired.
5. We need to show
$$\begin{aligned}
&-\tau\cdot\log\left(\sum_{i=1}^{k}2\mu_i(k\ell)^\tau\cdot\prod_{j=i+1}^{k}d_j\right) = \omega(k\log\lambda)\\
\Longleftarrow\ &\tau\cdot(\log(1/\mu)-\log k-\tau\log(k\ell)-k\log d) = \omega(\log\log\log n\cdot\log n)\\
\Longleftarrow\ &(\log n)^{1/2}\cdot\left(\frac{\log n}{\log\log n}-o\left(\frac{\log n}{\log\log n}\right)\right) = \omega(\log\log\log n\cdot\log n),
\end{aligned}$$
as desired.
6. $\mathsf{iMatMul}$ can be computed in depth
$$O(\log n) = O(\log\lambda)$$
7. Reach can be computed in depth
$$O(\log n) = O(\log\lambda)$$
8. By Lemma 8, it follows from Assumption 2 (via a standard hybrid argument for multiple independent secrets, and viewing $\mathbf{M}$ as secrets for $\mathsf{RingLPN}^{s\sim\chi}$) that for every $i \in [k]$ that: For any polynomial $m = \mathsf{poly}(\lambda)$,
$$\mathbf{V}\cdot\mathbf{M}_i + \mathbf{E} \approx_{compt.} \mathbf{U},$$
where $\mathbf{V} \leftarrow\!\$\ \mathbb{F}_q^{m\times n}, \mathbf{M}_i \leftarrow\!\$\ \mathcal{M}_i \in \mathbb{F}_q^{n\times\lambda n}$, $\mathbf{E} \leftarrow\!\$\ \mathsf{Ber}_{\mu_i}^{m\times\lambda\cdot n}, \mathbf{U} \leftarrow\!\$\ \mathbb{F}_q^{m\times\lambda\cdot n}$.

□

7 Instantiation III: NC^1 PRFs from Quasi-Polynomial Hardness of LPN

7.1 The LPN Assumption

In this section, we recall the LPN assumption.

Definition 24 (LPN [BFKL93])**.** Let $n, m, t \in \mathbb{N}$, $\mu \in (0, 1/2), \epsilon \in (0, 1)$. The $[n, m, \mu, t, \epsilon]_2$-LPN assumption states that

$$(\mathbf{A}, \mathbf{s} \cdot \mathbf{A} + \mathbf{e}) \approx_{t,\epsilon} (\mathbf{A}, \mathbf{u}),$$

where $\mathbf{s} \leftarrow\$ \mathbb{F}_2^n$, $\mathbf{A} \leftarrow\$ \mathbb{F}_2^{n \times m}$, $\mathbf{e} \leftarrow\$ \mathsf{Ber}_\mu^m$, and $\mathbf{u} \leftarrow\$ \mathbb{F}_2^m$.

Assumption 3 (Quasi-polynomial hardness of LPN)**.** Let $n \in \mathbb{N}$. Let $m = m(n), \mu = \mu(n), t = t(n), \epsilon = \epsilon(n)$ all be functions of n. We say LPN is quasi-polynomially hard if for any noise rate

$$\mu(n) = \Omega(n^{-1/\log\log n}),$$ [22]

there exists constant $c_1, c_2, c_3 > 0$, such that for

$$m(n) = 2^{c_1 \log^{1.1} n}, \quad t(n) = 2^{c_2 \log^{1.1} n}, \quad \epsilon(n) = 2^{-c_3 \log^{1.1} n},$$

the $[n, m, \mu, t, \epsilon]_2$-LPN assumption (Definition 24) holds for all sufficiently large n.

Under a constant noise rate (i.e., $\mu = \Theta(1)$), the best known LPN solvers [BKW03, LF06] require time and query complexity both $2^{\Omega(n/\log n)}$. The time complexity goes up to $2^{\Omega(n/\log\log n)}$ when $m = \mathsf{poly}(n)$ [Lyu05], or even $2^{\Omega(n)}$ given only $m = O(n)$ samples [MMT11]. Under our slightly sub-polynomial noise rate $\mu = n^{-1/\log\log n}$, for super-polynomial m, the best known attacks [Ste89, CC98, BLP11, Kir11, BJMM12] are not asymptotically better, i.e., still at the order of $2^{\tilde{\Omega}(n)}$. The quasi-polynomial (or subexponential) LPN was used in several previous works [YZ16, YZW+19, JLS21, CM21, JLS22, RVV24].

7.2 Reduction to LPN with Regular Sparse Coefficients

We introduce a variant of LPN, called rsLPN, where each column of $\mathbf{A}$ is a regular d-sparse vector, i.e., concatenation of d 1-sparse vectors in $\mathbb{F}_2^{n/d}$.

Definition 25 (rsLPN)**.** Let $n, m, d, t \in \mathbb{N}, d < n$, $\mu \in (0, 1/2), \epsilon \in (0, 1)$. The $[n, m, \mu, d, t, \epsilon]_2$-rsLPN assumption states that

$$(\mathbf{A}, \mathbf{s} \cdot \mathbf{A} + \mathbf{e}) \approx_{t,\epsilon} (\mathbf{A}, \mathbf{u}),$$

where $\mathbf{s} \leftarrow\$ \mathbb{F}_2^n$, $\mathbf{A} \leftarrow\$ \mathbb{F}_2^{n \times m}$ such that each column of $\mathbf{A}$ is a random regular d-sparse vector, $\mathbf{e} \leftarrow\$ \mathsf{Ber}_\mu^m$, and $\mathbf{u} \leftarrow\$ \mathbb{F}_2^m$.

Lemma 12 (**Reduction from LPN to rsLPN** [BBTV25])**.** *Let $n, m, d, t \in \mathbb{N}, d < n$, $\mu \in (0, 1/2), \epsilon \in (0, 1)$. $[n, m, \mu, d, t, \epsilon]_2$-rsLPN holds if $[d\log(n/d), m, \mu, t + 5mnd\log(n/d), \epsilon]_2$-LPN holds.*

We next show how to choose a sequence of parameters that reduces the quasi-polynomial hardness of a sequence of rsLPN instances to the quasi-polynomial hardness of LPN.

[22] We note that this noise level is only slightly sub-constant, and larger than any inverse polynomial

Definition 26 (Quasi-polynomial hardness of (a sequence of) rsLPN)**.** Let $n \in \mathbb{N}$. Let $m = m(n), \mu = \mu(n), d = d(n), t = t(n), \epsilon = \epsilon(n)$ all be functions of $n \in \mathbb{N}$. We say that rsLPN is quasi-polynomially hard if for a sequence of integers $i = 1, ..., \log\log\log n$, for any

$$\mu_i(n) = \Theta(n^{-1/(\log\log n)^{4i+2}}), \quad d_i(n) = \Theta(n^{1/(\log\log n)^{4i}}),^{23}$$

there exists constant $c_1, c_2, c_3 > 0$, such that for

$$m_i(n) = 2^{c_1 \log^{1.01} n}, \quad t_i(n) = 2^{c_2 \log^{1.01} n}, \quad \epsilon_i(n) = 2^{-c_3 \log^{1.01} n},$$

the $[n, m_i, \mu_i, d_i, t_i, \epsilon_i]_2$-rsLPN assumption holds for all sufficiently large n.

The following lemma follows by simply plugging in the said parameters in Lemma 12.

Lemma 13. *If* LPN *is quasi-polynomially hard (Assumption 3), then* rsLPN *is also quasi-polynomially hard (Definition 26).*

Proof. See Appendix A.6. □

7.3 Configuration

Configuration 3 (NC^1 PRFs from quasi-poly LPN)**.** Let λ be a security parameter.

1. Dimension $n = \lambda^{1/\log\log\lambda}$, modulus $q = 2$.
2. Maximum recursion depth $\tau = (\log\lambda)^{1/2}$.
3. Number of input chunks $k = \log\log\log\lambda$.
   (a) For $i \in [k]$, noise rate
   $$\mu_i = \Theta(n^{-1/(\log\log n)^{4i+2}})$$
   (chosen to be an inverse power of two). Let $\ell_i = \log(1/\mu_i)$. Let Bernoulli sample function $\mathsf{S\text{-}Ber}_{\mu_i} : \mathbb{F}_2^{\ell_i} \to \mathbb{F}_2$: $\mathsf{S\text{-}Ber}_{\mu_i}(\mathbf{x}) := \prod_{j=1}^{\ell_i} x_j$.
   (b) For $i \in [k]$, sparsity
   $$d_i = n^{1/(\log\log n)^{4i}},$$
   and distribution $\mathcal{M}_i$ is a random regular d_i-sparse matrix in $\{0,1\}^{n \times n\lambda}$.

Corollary 3 (NC^1 **KH-PRFs from quasi-polynomial** LPN)**.** *Assuming the quasi-polynomial hardness of* LPN *(Assumption 3), there exists a secure KH-PRF family in* NC^1*.*

Proof. It suffices to show that Configuration 3 is a valid key-homomorphic configuration (Definition 14), then the corollary follows from Theorem 6.

1. $k = \log\log\log\lambda = \omega(1)$, as desired.
2. $\log k + \log\log q = O((\log\lambda)/\tau)$, as desired.

[23] It follows that $\mu_i \cdot d_i \gg 1$.

3. For $i \in [k]$, $\mathsf{S\text{-}Ber}_{\mu_i}$ can be computed in depth

$$O(\log \ell_i) = O(\log \log(1/\mu_i)) = O(\log \log n - \log \log \log^2 n) = O(\log \log n) = O((\log \lambda)/\tau),$$

as desired.

4. We need to show

$$\begin{aligned} & \min(k, \tau) \cdot \sum_{i=1}^{k} \log d_i = O(\log \lambda) \\ \Longleftarrow\ & \log \log \log \lambda \cdot \log \log \log \lambda \cdot \frac{\log n}{\log \log^4 n} = O(\log \lambda) \\ \Longleftrightarrow\ & \log \log \log^2 \lambda \cdot \frac{1}{\log \log^5 \lambda} = O(1), \end{aligned}$$

as desired.

5. We need to show

$$-\tau \cdot \log \left(\sum_{i=1}^{k} 2\mu_i (k\ell)^{\tau} \cdot \prod_{j=i+1}^{k} d_j \right) = \omega(k \log \lambda)$$

Denote $T_i = 1/(2\mu_i (k\ell)^{\tau} \cdot \prod_{j=i+1}^{k} d_j)$. Therefore,

$$\begin{aligned} \log T_i &= \log(1/2\mu_i) - \tau \log(k\ell) - \sum_{j=i+1}^{k} \log d_j \\ &= \Omega \left(\frac{\log n}{\log \log^{4i+2} n} - \log^{1/2} \lambda \cdot \log(\log \log \log \lambda \cdot \log n) - \sum_{j=i+1}^{k} \frac{\log n}{\log \log^{4j} n} \right) \\ &= \Omega \left(\frac{\log n}{\log \log^{4i+2} n} - k \cdot \frac{\log n}{\log \log^{4i+4} n} \right) \\ &= \Omega \left(\frac{\log n}{\log \log^{4i+2} n} \right) \end{aligned}$$

It then suffices to show

$$\begin{aligned} & \tau \cdot \log \left(\frac{1}{\sum_{i=1}^{k} 1/T_i} \right) = \omega(k \log \lambda) \\ \Longleftarrow\ & \tau \cdot (\log T_k - \log k) = \omega(k \log \lambda) \\ \Longleftarrow\ & \log^{1/2} \lambda \cdot \left(\frac{\log \lambda}{\log \log^{4 \log \log \log \lambda + 2 + 1} n} - \log k \right) = \omega(\log \log \log \lambda \cdot \log \lambda) \end{aligned}$$

as desired.

6. $\mathsf{iMatMul}$ can be computed (by vanilla method) in depth

$$O\left(\log k \cdot \log n\right) = O(\log \log \log \log \lambda \cdot \frac{\log \lambda}{\log \log \lambda}) = O(\log \lambda),$$

as desired.

7. Reach can be computed (by first computing $\mathsf{iMatMul}$ over $\mathbb{Z}_q$ for $q = 1 + \prod_{i=1}^{k} d_i$ (then, all intermediate values are below q), then check the nonzero entries) in depth

$$O\left(\log k \cdot (\log n + \log\log q)\right) = O(\log \lambda)$$

8. By Lemma 13, it follows from Assumption 3 (via a standard hybrid argument for multiple independent secrets) that for every $i \in [k]$: For any polynomial $m = \mathsf{poly}(\lambda)$,

$$(\mathbf{M}_i, \mathbf{V} \cdot \mathbf{M}_i + \mathbf{E}) \approx_{compt.} (\mathbf{M}_i, \mathbf{U}),$$

where $\mathbf{V} \leftarrow\!\$\, \mathbb{F}_q^{m \times n}, \mathbf{M}_i \leftarrow\!\$\, \mathcal{M}_i \in \mathbb{F}_q^{n \times \lambda n}$, $\mathbf{E} \leftarrow\!\$\, \mathsf{Ber}_{\mu_i}^{m \times \lambda \cdot n}$, $\mathbf{U} \leftarrow\!\$\, \mathbb{F}_q^{m \times \lambda \cdot n}$.

□

Acknowledgements

Ding and Komargodski were supported in part by a grant from the Israel Science Foundation (ISF Grant No. 1774/20), and by the European Union (ERC, SCALE,101162665). Jain was supported in part by a Google Faculty Research Scholarship, Amazon, Cylab and QCIT of CMU, 0xParc, a Stellar Foundation Research Grant, and the NSF career award. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

A Omitted Proofs

A.1 Proof of Lemma 3

Lemma 14 (restating Lemma 3). *Let all parameters be as in Definition 5-9. Let $\mathbb{F} = \mathbb{F}_q$. Every operation in Definition 5-9 can be done in $O(\log n + \log\log q)$ depth.*

Proof. All selection operations are computable in depth $O(\log n)$.

Let w be a padding parameter. For support of a vector Supp, we construct the circuit as follows. On input $\mathbf{v} \in \mathbb{F}_q^n$:

1. $a_i \leftarrow \mathbb{I}[\mathbf{v}_i > 0] \in \{0,1\}$ for $i \in [n]$ in parallel.
2. $b_i \leftarrow \sum_{j=1}^{i} a_i \in [n]$ (addition over integer) for $i \in [n]$ in parallel.
3. For $j \in [w]$ in parallel:
   (a) For $i \in [n]$ in parallel:
   $$c_i = \begin{cases} \mathbb{I}[b_i = j], & j = 1, \\ \mathbb{I}[b_i = j \wedge \mathbf{b}_{i-1} = j-1] & j > 1. \end{cases}$$
   (b) Compute $p \leftarrow \sum_{i=1}^{n} c_i \cdot i$.
   (c) $p_j \leftarrow p \in [n]$
4. Output $p_1, ..., p_w$.

The first step takes depth $\log\log q$, other steps take depth $O(\log n)$. The overall depth is $O(\log\log q + \log n)$. $\mathsf{RowSupp}$ can be computed by computing Supp for each column in parallel. □

A.2 Proof of Theorem 7

Theorem 11 (restating Theorem 7). Eval *computes* PRF *(Definition 15). That is, for every* $\mathbf{k} \in \mathcal{K}, x \in \mathcal{X}$, $\mathsf{Eval}_{\mathbf{k}}(x) = \mathsf{PRF}_{\mathbf{k}}(x)$.

Proof. Fix any PRF secret key. We introduce a sequence of hybrids as follows.

- $\mathsf{PRF}_{\mathbf{k}}$ (output restriction): This overloads the PRF defined in Definition 15. In addition to x, it takes an index set $\mathcal{I} \subseteq [n]$ and returns the restriction $\mathsf{PRF}(x)|_{\mathcal{I}} \in \mathbb{F}_2^{|\mathcal{I}|}$, instead of the full vector $\mathsf{PRF}(x) \in \mathbb{F}_2^n$. In particular, $\mathsf{PRF}(x) = \mathsf{PRF}(x, \{1, \ldots, n\})$.[24]
- $\mathsf{PRF}'_{\mathbf{k}}$: This is the same as the above PRF except that instead of calling F and selects the output based on $\mathcal{I}$ via $\mathsf{PRF}(x, \mathcal{I}) \coloneqq \mathsf{ColSelect}(\mathsf{F}(x, 0), \mathcal{I})$, it directly calls F' (Figure 7) to output $\mathsf{PRF}'(x, \mathcal{I}) \coloneqq \mathsf{F}'(x, \mathcal{I}, 0)$.

[24] We abuse notation and write PRF for both interfaces; the intended meaning is clear from whether a second input is present.

- $\mathsf{Eval}_{\mathbf{k}}$: This is the evaluation algorithm of our PRF construction (Construction 1).

Lemma 15. $\mathsf{PRF}_{\mathbf{k}} \equiv \mathsf{PRF}'_{\mathbf{k}}$.

Proof. This follows from the linearity of vector-matrix multiplication and the way we extend $\mathsf{S\text{-}Ber}$ to handle matrix input (i.e., operating on each column separately and independently). In more detail, take a closer look at the execution of F underlying PRF:

1. Since we only need to return $\mathbf{y}|_{\mathcal{I}}$, we do not need to compute $\mathbf{y}|_{[n]\setminus\mathcal{I}}$ at all.
2. Observe that $\mathbf{y}|_{\mathcal{I}} = \mathbf{v}\cdot\mathbf{M}|_{\mathcal{I}} + \sum \mathbf{e}\cdot\mathbf{M}|_{\mathcal{I}}$, and because the matrices $\mathbf{M}$ arising in the computation are sparse, each term only depends on $\mathbf{e}|_{\mathcal{I}'}$, where $\mathcal{I}'$ is the row support of $\mathbf{M}|_{\mathcal{I}}$. Now that we only need to know $\mathbf{e}|_{\mathcal{I}'}$, which is actually equal to $\mathsf{S\text{-}Ber}(\mathbf{R})|_{\mathcal{I}'}$ where $\mathbf{R}$ was the output returned by its invoked recursive call. Since $\mathsf{S\text{-}Ber}$ operates on each column of $\mathbf{R}$ independently, we only need $\mathbf{R}|_{\mathcal{I}'}$. In more detail, it holds that for any $\mathcal{I}$, we have
$$\mathsf{ColSelect}(\mathsf{S\text{-}Ber}(\mathbf{R}),\ \mathcal{I}) = \mathsf{S\text{-}Ber}(\mathsf{ColSelect}(\mathbf{R},\ \mathcal{I})),$$
by definition of how we extend $\mathsf{S\text{-}Ber}$ to handle matrix input (i.e., operating on each column separately and independently).
3. $\mathbf{r}$ (row of $\mathbf{R}$) is returned by a recursive call, and we are only interested in entries in $\mathcal{I}'$. This leads to our recursive definition of F' underlying PRF'.

□

Lemma 16. $\mathsf{PRF}'(x, \{1, ..., n\}) \equiv \mathsf{Eval}(x)$, *for any* $x \in [\lambda]^k$.

Proof. It follows from $\mathsf{F}'(x, \{j\}, 0) = \mathsf{G}(x, \{j\}, 0)$[25] for each $j \in [n]$, which we will prove.

Fix j. We compare the computation transcript between them. For any node in the recursion tree, which corresponds to an execution of $\mathsf{F}'_{(i)}(x_{1..i}, \mathcal{I}'_i, \mathsf{layer})$ (or $\mathsf{G}_{(i)}(x_{1..i}, \mathcal{I}_i, \mathsf{layer})$). We first prove the following lemma.

Lemma 17. *For any node at recursion tree, which induces a set $\mathcal{I}'_i$ (corresponds to the input to a call of $\mathsf{F}'_{(i)}$ at the node) and a set $\mathcal{I}_i$ (corresponds to $\mathsf{G}_{(i)}$). It holds that $\mathcal{I}'_i \subseteq \mathcal{I}_i$.*

Proof. We prove by induction on recursion depth $\mathsf{layer} = 0, 1, ..., \tau$.

- Base case: $\mathsf{layer} = 0$[26]. It holds that $\mathcal{I} = \mathcal{I}' = \{j\}$. The base case holds.
- Inductive step: for $\mathsf{layer} \in \{1, ..., \tau\}$. Assume the lemma holds for $\mathsf{layer} - 1$. Fix any node at depth layer of the recursion tree. Denote the corresponding execution by $\mathsf{F}'_{(i)}(x_{1..i}, \mathcal{I}'_i, \mathsf{layer})$ and $\mathsf{G}_{(i)}(x_{1..i}, \mathcal{I}_i, \mathsf{layer})$. We need to prove that $\mathcal{I}'_i \subseteq \mathcal{I}_i$. Denote their caller's by the execution of $\mathsf{F}'_j(x_{1..k'}, \mathcal{I}'_{k'}, \mathsf{layer} - 1)$ and $\mathsf{G}_{k'}(x_{1..k'}, \mathcal{I}_{k'}, \mathsf{layer} - 1)$, respectively. Note that $j \geq i$. The inductive assumption implies that $\mathcal{I}'_{k'} \subseteq \mathcal{I}_{k'}$.

[25] Note that the execution of G is parameterized by a precomputed $\mathcal{I}_i$ that depends on j. We explicitly include the $\mathcal{I}_i$ as part of the input of G for reference.

[26] Note that this base case of the induction should not be confused with the base cases of our recursion.

Now take a closer look at how we derive $\mathcal{I}_i$ from $\mathcal{I}_{k'}$ at the execution of the caller $\mathsf{G}_{(k')}$. Recall that they are derived from walking on the $(k+1)$-layered graph induced by $\mathbf{M}_{x_k}, ..., \mathbf{M}_{x_1}$. Specifically, $u \in \mathcal{I}_i$ if and only if node u at $i+1$-th layer of the layered graph is reachable if we start walking at nodes set $\mathcal{I}_{k'}$ at the k'-layer of the layered graph.

Now take a closer look at how we derive $\mathcal{I}'_i$ from $\mathcal{I}'_{k'}$ at the execution of the caller $\mathsf{F}''_{(k')}$. It holds that

$$\mathcal{I}_i \leftarrow \mathsf{RowSupp}(\mathsf{ColSelect}(\mathbf{M}_{(i+1,k')}, \mathcal{I}_{k'});\ w_i) \in [n]^{w_i},$$

Since $\mathbf{M}_{(i+1,k')} = \prod_{j=i+1}^{k'} \mathbf{M}_{x_j}$, observe that $v \in \mathcal{I}_i$ *only if* v is reachable if we walk from nodes in $\mathcal{I}_{k'}$ over the layered graph induced by $\mathbf{M}_{x_{k'}}, \dots \mathbf{M}_{x_i+1}$. Since $\mathcal{I}'_{k'} \subseteq \mathcal{I}_{k'}$ and reachability is a monotone function, therefore $\mathcal{I}'_i \subseteq \mathcal{I}_i$.

□

We are now ready to prove the lemma by induction on recursion depth $\mathsf{layer} = \tau, \tau-1, ..., 0$.

Lemma 18. *For any node at recursion tree, denote the corresponding execution instance by* $\mathsf{F}'_{(i)}(x_{1..i}, \mathcal{I}'_i, \mathsf{layer})$ *and* $\mathsf{G}_{(i)}(x_{1..i}, \mathcal{I}_i, \mathsf{layer})$*, respectively. Denote their return value by* $\mathbf{y}' \in \mathbb{F}_2^{|\mathcal{I}'_i|}$*, and* $\mathbf{y} \in \mathbb{F}_2^{|\mathcal{I}_i|}$ *respectively. It holds that* $\mathbf{y}|_{\mathcal{I}'_i} = \mathbf{y}'$*.*

Proof. We prove by induction on recursion depth $\mathsf{layer} = \tau, \tau-1, ..., 0$.

- Base case: $\mathsf{layer} = \tau$. The base layer always outputs a zero vector. Therefore $\mathbf{y}' = \mathsf{F}'_{(i)}(x_{1..i}, \mathcal{I}'_i, \tau) = \mathbf{0}$, and $\mathbf{y} = \mathsf{G}_{(i)}(x_{1..i}, \tau) = \mathbf{0}$. The base case holds.
- Inductive step: for $\mathsf{layer} \in \{\tau-1, \dots, 0\}$, assume the lemma holds for $\mathsf{layer}+1$. Fix any node at depth layer of the recursion tree. Denote the corresponding execution by $\mathsf{F}'_{(i)}(x_{1..i}, \mathcal{I}'_i, \mathsf{layer})$ and $\mathsf{G}_{(i)}(x_{1..i}, \mathcal{I}_i, \mathsf{layer})$. By Lemma 17, it holds that $\mathcal{I}'_i \subseteq \mathcal{I}_i$. Therefore, in the execution of $\mathsf{G}_{(i)}(x_{1..i}, \mathcal{I}_i, \mathsf{layer})$, we only need to care about its output in a smaller subset $\mathcal{I}'_i$. Then by the same argument as in Lemma 15 (namely, by the linearity of vector matrix multiplication, and by the fact that the $\mathsf{S\text{-}Ber}$ operates on each column independently), in order to compute that, we only rely on the $\mathbf{y}_{(j)}|_{\mathcal{I}'_j}$ returned by its recursive call $\mathsf{G}_{(j)}$ for some $j \le i$. By our inductive assumption, $\mathbf{y}_{(j)}|_{\mathcal{I}'_j}$ agrees with $\mathbf{y}'_{(j)}|_{\mathcal{I}'_j}$, which concludes the proof of inductive step.

□

Note that when $\mathsf{layer} = 0$, we have $\mathcal{I}_k = \mathcal{I}'_k = \{j\}$, Lemma 18 implies that $\mathsf{F}'_{(k)}(x, \{j\}, 0) = \mathsf{G}_{(k)}(x, \{j\}, 0)$, which concludes the proof of Lemma 16. □

This concludes the correctness proof. □

A.3 Proof of Theorem 9

Theorem 12 (restating Theorem 9). *Let* $t, \epsilon_1, \epsilon_2 > 0$*,* $Q \le \lambda^k$*. If for every* $i \in [k]$*,*

$$\mathbf{V} \cdot \mathbf{M}_i + \mathbf{E} \approx_{t,\epsilon} \mathbf{U},$$

where $\mathbf{V} \leftarrow\!\$\, \mathbb{F}_q^{Q\times n}, \mathbf{M}_i \leftarrow\!\$\, \mathcal{M}_i \in \mathbb{F}_q^{n\times \lambda n}$, $\mathbf{E} \leftarrow\!\$\, \mathsf{Ber}_{\mu_i}^{Q\times \lambda\cdot n}, \mathbf{U} \leftarrow\!\$\, \mathbb{F}_q^{Q\times \lambda\cdot n}$. *Then,* $\mathsf{PRF}_{\mathbf{k}}\colon \{0,1\}^{k\log\lambda} \to \{0,1\}^n$ *defined in Definition 15 is a*

$$(Q, t - \mathsf{poly}(\lambda), \tau k^2\cdot\epsilon + \lambda^k\cdot n\cdot\epsilon')$$

-pseudorandom function (Definition 3), where

$$\epsilon' \le \left(\sum_{i=1}^{k} 2\mu_i(k\ell)^\tau \cdot \prod_{j=i+1}^{k} d_j\right)^\tau$$

Proof. We define hybrid games as follows.

Hybrid H_0**.** This is the real PRF game: the attacker has oracle access to $\mathsf{PRF}_{\mathbf{k}}(\cdot)$ for $\mathbf{k} \leftarrow\!\$\, \mathcal{K}$.

Hybrid H_1**.** This is the same as Hybrid H_0 except that the attacker has oracle access to $\widetilde{\mathsf{PRF}}_{\mathbf{k}}(\cdot)$, which is defined the same as $\mathsf{PRF}_{\mathbf{k}}$, except that instead of invoking F, it will invoke $\widetilde{\mathsf{F}}$ defined in Figure 9.

Hybrid H_2**.** This is the same as Hybrid H_1 except that instead of invoking $\widetilde{\mathsf{F}}$, $\widetilde{\mathsf{PRF}}_{\mathbf{k}}$ will invoke $\widetilde{\mathsf{G}}$ defined in Figure 10 (invoke for each output bit as we did in Construction 1).

Hybrid H_3**.** This is the ideal PRF game: the attacker has oracle access to a random function $R(\cdot)$.

As a corollary of Theorem 7, it follows that $H_1 \equiv H_2$, since they are just two different ways to compute the same function. It suffices to show that $H_0 \approx_{stat.} H_2$ (resilience property), $H_1 \approx_{compt.} H_3$ (GGM-like argument).

Lemma 19. *For any adversary* $\mathcal{A}$*, we have*

$$\mathsf{Adv}_{\mathcal{A}}(H_0, H_2) \le \lambda^k\cdot n\cdot\left(\sum_{i=1}^{k} 2\mu_i(k\ell)^\tau \cdot \prod_{j=i+1}^{k} d_j\right)^\tau. \tag{11}$$

Proof. We lower bound the probability that the $\mathsf{PRF}_{\mathbf{k}}$ in H_0 and $\widetilde{\mathsf{PRF}}_{\mathbf{k}}$ H_2 have the same truth table over the randomness of secret key. Then, the failure probability upper bounds any attacker's distinguishing advantage. Therefore, it suffices to prove the following claim. The lemma then follows by applying a union bound over all $x \in [\lambda]^k$.

Claim 1. *Fix any* $x \in [\lambda]^k$*, it holds that*

$$\Pr_{\mathbf{k}}\left[\mathsf{PRF}_{\mathbf{k}}(x) \ne \widetilde{\mathsf{PRF}}_{\mathbf{k}}(x)\right] \le n\cdot\left(\sum_{i=1}^{k} 2\mu_i(k\ell)^\tau \cdot \prod_{j=i+1}^{k} d_j\right)^\tau,$$

where the probability is over the randomness of the secret key $\mathbf{k}$*.*

Proof. Fix any $x \in \{0,1\}^{k\log n}$. Fix any output index $j \in [n]$. We consider the transcript of the intermediate results during the evaluation of $\mathsf{PRF}_{\mathbf{k}}(x)$ and $\widetilde{\mathsf{PRF}}_{\mathbf{k}}(x)$ for computing the j-th output bit.

The evaluation of $\mathsf{PRF}_{\mathbf{k}}(x)$ (for computing the j-th output bit) is a bunch of recursive calls of G. For a recursion depth $\alpha \in [\tau]$, we collect all the return values of $\mathsf{G}_\star(\star, \mathsf{layer} = \alpha)$, and denote by a

matrix $\mathbf{R}^{(\alpha)} \in \mathbb{F}_q^{T_\alpha \times w_\star}$, where $T_\alpha \le T$. Similarly, we collect all padding vectors that are used at recursion depth α, denoted by $\mathbf{P}^{(\alpha)} \in \mathbb{F}_q^{T_\alpha \times w_\star}$. We collect all vectors $\overline{\mathbf{y}}$ that are padded, denoted by $\mathbf{Y}^{(\alpha)} \in \mathbb{F}_q^{T_\alpha \times w_\star}$. The same notation applies for $\widetilde{\mathsf{PRF}}_{\mathbf{k}}(x)$, analogously.

Observe that if $\mathsf{PRF}_{\mathbf{k}}(x)|_j \neq \widetilde{\mathsf{PRF}}_{\mathbf{k}}(x)|_j$, then it must hold that for all $\alpha \in [\tau]$, $\mathbf{R}^{(\alpha)} \neq \widetilde{\mathbf{R}}^{(\alpha)}$. This is due to the fact that the execution of their callers is deterministic. In other words, if there exists α, such that $\mathbf{R}^{(\alpha)} = \widetilde{\mathbf{R}}^{(\alpha)}$, then it follows that $\mathbf{R}^{(\alpha-1)} = \widetilde{\mathbf{R}}^{(\alpha-1)}$ since $\mathsf{F}_{(\star)}(\star, \mathsf{layer} = \alpha - 1)$ is a deterministic function given the value $\mathbf{R}^{(\alpha)}$ returned by its invoked recursive calls. Repeating this argument yields $\mathsf{PRF}_{\mathbf{k}}(x)|_j = \widetilde{\mathsf{PRF}}_{\mathbf{k}}(x)|_j$.

For $\alpha = 1, \ldots, \tau$, we consider the probability of $\mathbf{R}^{(\alpha)} \neq \widetilde{\mathbf{R}}^{(\alpha)}$,

$$\begin{aligned}
\Pr_{\mathbf{k}}[\mathbf{R}^{(\alpha)} \neq \widetilde{\mathbf{R}}^{(\alpha)}] &= \Pr[\mathsf{S\text{-}Ber}(\mathbf{R}^{(\alpha+1)}) \neq \mathsf{S\text{-}Ber}(\widetilde{\mathbf{R}}^{(\alpha+1)})] \\
&= \Pr[\mathsf{S\text{-}Ber}(\mathbf{P}^{(\alpha+1)} + \mathbf{Y}^{(\alpha+1)}) \neq \mathsf{S\text{-}Ber}(\mathbf{P}^{(\alpha+1)} + \widetilde{\mathbf{Y}}^{(\alpha+1)})] \\
&\le \sum_{i=1}^{k} 2\mu_i(k\ell)^\tau \cdot w_i,
\end{aligned} \tag{12}$$

where the last line is due to applying a union bound on Lemma 1 and the fact that $\mathbf{P}^{(\alpha+1)}$ is uniformly random. Therefore, it holds that

$$\begin{aligned}
\Pr_{\mathbf{k}}\left[\mathsf{PRF}_{\mathbf{k}}(x)|_j \neq \widetilde{\mathsf{PRF}}_{\mathbf{k}}(x)|_j\right] &= \Pr\left[\bigcap_{\alpha=1}^{\tau} \mathbf{R}^{(\alpha)} \neq \widetilde{\mathbf{R}}^{(\alpha)}\right] \\
&= \prod_{\alpha=0}^{\tau-1} \Pr\left[\mathbf{R}^{(\alpha)} \neq \widetilde{\mathbf{R}}^{(\alpha)} \mid \mathbf{R}^{(\alpha+1)} \neq \widetilde{\mathbf{R}}^{(\alpha+1)}\right] \\
&\le \prod_{\alpha=0}^{\tau-1} \Pr\left[\mathbf{R}^{(\alpha)} \neq \widetilde{\mathbf{R}}^{(\alpha)}\right] \\
&\le \left(\sum_{i=1}^{k} 2\mu_i(k\ell)^\tau \cdot \prod_{j=i+1}^{k} d_j\right)^\tau.
\end{aligned} \tag{13}$$

Applying a union bound over all output indices $j \in [n]$ yields the claim.

□

□

Lemma 20. *For any efficient adversary $\mathcal{A}$ making at most Q queries, we have*

$$\mathsf{Adv}_{\mathcal{A}}(H_1, H_3) \le \tau \cdot k^2 \cdot \epsilon. \tag{14}$$

Proof. We define a sequence of intermediate hybrid games H_{1,α^*} for $\alpha^* \in [\tau]$ as follows:

Hybrid H_{1,α^*}, $\alpha \in [\tau]$: This is the same as Hybrid H_1 except that instead of invoking $\widetilde{\mathsf{F}}$, it will invoke $\widetilde{\mathsf{F}}^{(1,\alpha^*)}$ defined in Figure 15.

It follows by definition that $H_{1,\tau} = H_1$, $H_{1,0} = H_3$. It suffices to prove the following lemma.

Hybrid H_{1,α^*}: $\widetilde{\mathsf{F}}$

Global Parameters: Max recursion depth τ, iterators, denoted by $\mathsf{Iter}_{(i)}$ for $i \in [k]$, and

$$(\mathbf{M}_{1,1}, \ldots, \mathbf{M}_{k,\lambda}), \text{ where } \mathbf{M}_{i,x} \in \mathbb{F}_q^{n \times n} \text{ for } i, x \in [k] \times [\lambda], \qquad \mathbf{V}_{(i)},\ \mathbf{P}_{(i)} \in \mathbb{F}_q^{T \times n}, \text{ for } i \in [k]$$

For each input length $i \in [k]$.

$$R_{(i)} = (R_1, ..., R_T), \text{ where each } R_j \text{ is a random function } R_j \colon [\lambda]^i \to \mathbb{F}_q^n$$

For every $k' = 1, \ldots, k$, define $\widetilde{\mathsf{F}}_{(k')}$ as follows.

Function $\widetilde{\mathsf{F}}_{(k')}(x \in [\lambda]^{k'},\ \mathsf{layer})$:

1. If $\mathsf{layer} = 0$, initialize an iterator $\mathsf{Iter}_{(k')}.\mathsf{Init}(\mathbf{V}_{(k')}, \mathbf{P}_{(k')}, R_{(k')})$.
2. If $\mathsf{layer} = \alpha^*$, return $R(x) \in \mathbb{F}_q^n$, where $R \leftarrow \mathsf{Iter}'_{(k')}.\mathsf{Next}()$.
3. $(\mathbf{v},\ \mathbf{p}) \leftarrow \mathsf{Iter}_{(k')}.\mathsf{Next}() \in \mathbb{F}_q^n \times \mathbb{F}_q^n$.
4. For each input prefix $i \in [k']$:

$$\mathbf{e}_i \leftarrow \mathsf{S\text{-}Ber}_{\mu_i}(\mathbf{R}_i) \in \mathbb{F}_q^n, \quad \text{where} \quad \mathbf{R}_i \ \leftarrow \widetilde{\mathsf{F}}^{\ell_i}_{(i)}(x_{1,..,i},\ \mathsf{layer}{+}1) \in \mathbb{F}_q^{\ell_i \times n}.$$

5. Compute:

$$\mathbf{y} \ \leftarrow \ \mathbf{v} \prod_{i=1}^{k'} \mathbf{M}_{i,x_i} + \sum_{i=1}^{k'} \mathbf{e}_i \cdot \prod_{j=i+1}^{k'} \mathbf{M}_{j,x_j} \in \mathbb{F}_q^n,$$

6. Output $\mathbf{y} + \mathbf{p} \in \mathbb{F}_q^n$.

Figure 15: Hybrid H_{1,α^*}.

Lemma 21. *For any efficient adversary $\mathcal{A}$ making at most Q queries, we have*

$$\mathsf{Adv}_{\mathcal{A}}(H_{1,\alpha^*}, H_{1,\alpha^*}) \leq k^2 \cdot \epsilon. \tag{15}$$

Proof. We define a sequence of intermediate hybrid games as follows:

Hybrid G_0. This is the real PRF game: the attacker has oracle access to $G_{\mathbf{k}}(\cdot)$ for $\mathbf{k} \leftarrow\!\!\$\ \mathcal{K}$, which is defined as follows: On input $x \in [\lambda]^k$,

1. For each input prefix $i \in [k]$:

$$\mathbf{e}_i \leftarrow \mathsf{S\text{-}Ber}_{\mu_i}(\mathbf{R}_i) \in \mathbb{F}_q^n, \quad \text{where} \quad \mathbf{R}_i \ \leftarrow R(x_{1,..,i}) \in \mathbb{F}_q^{\ell_i \times n}.$$

2. Compute:

$$\mathbf{y} \leftarrow \mathbf{v} \prod_{i=1}^{k} \mathbf{M}_{i,x_i} + \sum_{i=1}^{k} \mathbf{e}_i \cdot \prod_{j=i+1}^{k} \mathbf{M}_{j,x_j} \in \mathbb{F}_q^n,$$

3. Output $\mathbf{y} + \mathbf{p} \in \mathbb{F}_q^n$.

Hybrid G_{1,i^*} for $i^* \in [k+1]$. the attacker has oracle access to $G_{\mathbf{k}}^{(i^*)}(\cdot)$ for $\mathbf{k} \leftarrow\!\!\$\, \mathcal{K}$, which is defined as follows: On input $x \in [\lambda]^k$,

1. For each input prefix $i = i^*, \ldots, k$:

$$\mathbf{e}_i \leftarrow \mathsf{S\text{-}Ber}_{\mu_i}(\mathbf{R}_i) \in \mathbb{F}_q^n, \quad \text{where} \quad \mathbf{R}_i \leftarrow R(x_{1,..,i}) \in \mathbb{F}_q^{\ell_i \times n}.$$

2. If $i^* = 1$, $\mathbf{v}_{(0)} \leftarrow \mathbf{v}$. Else, $\mathbf{v}_{(i^*-1)} \leftarrow R'(x_{1,\ldots,i^*-1})$.

3. Compute:

$$\mathbf{y} \leftarrow \mathbf{v}_{(i^*-1)} \prod_{i=i^*}^{k} \mathbf{M}_{i,x_i} + \sum_{i=i^*}^{k} \mathbf{e}_i \cdot \prod_{j=i+1}^{k} \mathbf{M}_{j,x_j} \in \mathbb{F}_q^n,$$

4. Output $\mathbf{y} + \mathbf{p} \in \mathbb{F}_q^n$.

Hybrid G_2. This is the ideal PRF game: the attacker has oracle access to a random function $R(\cdot)$.

It suffices to prove that $\mathsf{Adv}_{\mathcal{A}}(G_0, G_1) \leq k \cdot \epsilon$, then for hybrid H_{1,α^*} replacing $\widetilde{\mathsf{F}}_{(k')}$ for $k' \in [k]$ at layer α with $R_{(k')}(\cdot)$ yields the lemma.

It follows by definition that $G_0 = G_{1,1}$, $G_{1,k+1} = G_2$. It suffices to prove that

$$\mathsf{Adv}_{\mathcal{A}}(G_{1,i^*}, G_{1,i^*+1}) \leq \epsilon.$$

Let $\mathcal{A}$ be a distinguisher for the Games G_{1,i^*} and G_{1,i^*+1} that makes at most Q queries. We construct an algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to distinguish $\mathbf{V} \cdot \mathbf{M}_{i^*} + \mathbf{E}$ from random with the same advantage, a contradiction.

Algorithm $\mathcal{B}$ receives as input $\mathbf{B} \in \mathbb{F}_q^{Q \times \lambda \cdot n}$. It parses $\mathbf{B} = (\mathbf{B}_{i,j})_{i \in [Q], j \in [\lambda]}$ where $\mathbf{B}_{i,j} \in \mathbb{F}_q^{1 \times n}$. It answers $\mathcal{A}$'s query x as follows.

1. It sets $\mathbf{v}_{(i^*)} \leftarrow \mathbf{B}_{idx(x_{1\ldots i^*-1}), x_{i^*}}$, where the index $idx = idx(x_{1\ldots i^*-1}) \in [Q]$ is chosen to ensure consistency among $\mathcal{A}$'s queries. For example it can be set to the smallest previously unused index when $x_{1\ldots i^*-1}$ occurs for the first time.

2. It computes

$$\mathbf{y} \leftarrow \mathbf{v}_{(i^*)} \prod_{i=i^*+1}^{k} \mathbf{M}_{i,x_i} + \sum_{i=i^*+1}^{k} \mathbf{e}_i \cdot \prod_{j=i+1}^{k} \mathbf{M}_{j,x_j} \in \mathbb{F}_q^n,$$

3. Output $\mathbf{y} + \mathbf{p} \in \mathbb{F}_q^n$.

Then we show that $\mathcal{B}$ either simulates G_{1,i^*} or G_{1,i^*+1}.

- If $\mathcal{B}$ receives $\mathbf{V} \cdot \mathbf{M}_{i^*} + \mathbf{E}$ for $\mathbf{V} \leftarrow\$ \mathbb{F}_q^{Q \times n}, \mathbf{M}_{i^*} \leftarrow\$ \mathcal{M}_{i^*} \in \mathbb{F}_q^{n \times \lambda n}$, $\mathbf{E} \leftarrow\$ \mathsf{Ber}_{\mu_{i^*}}^{Q \times \lambda \cdot n}, \mathbf{U} \leftarrow\$ \mathbb{F}_q^{Q \times \lambda \cdot n}$. Then

$$\begin{aligned}
\mathbf{y} &= \mathbf{v}_{(i^*)} \prod_{i=i^*+1}^{k} \mathbf{M}_{i,x_i} + \sum_{i=i^*+1}^{k} \mathbf{e}_i \cdot \prod_{j=i+1}^{k} \mathbf{M}_{j,x_j} \\
&= (\mathbf{v}_{idx(x_{1\ldots,i^*-1})} \cdot \mathbf{M}_{i^*,x_{i^*}} + \mathbf{e}_{idx(x_{1\ldots,i^*-1}),x_{i^*}}) \prod_{i=i^*+1}^{k} \mathbf{M}_{i,x_i} + \sum_{i=i^*+1}^{k} \mathbf{e}_i \cdot \prod_{j=i+1}^{k} \mathbf{M}_{j,x_j} \\
&= (\mathbf{v}_{idx(x_{1\ldots,i^*-1})} \prod_{i=i^*}^{k} \mathbf{M}_{i,x_i} + \sum_{i=i^*}^{k} \mathbf{e}_i \cdot \prod_{j=i+1}^{k} \mathbf{M}_{j,x_j}
\end{aligned}$$

  Therefore, $\mathcal{B}$ perfectly simulates G_{1,i^*}.

- If $\mathcal{B}$ receives $\mathbf{U}$ for $\mathbf{U} \leftarrow\$ \mathbb{F}_q^{Q \times \lambda \cdot n}$, then $\mathbf{v}_{(i^*)} = \mathbf{U}_{idx(x_{1\ldots i^*-1}),x_{i^*}}$. Therefore $\mathcal{B}$ perfectly simulates G_{1,i^*+1}.

Therefore, $\mathcal{B}$ has the same distinguishing advantage as $\mathcal{A}$.

□

□

□

A.4 Proof of Lemma 8

Lemma 22 (restating Lemma 8). *Let all parameters be as in Definitions 20–22. The* $[n, m, \mu, t, \epsilon]_{\mathcal{R}_q}$-$\mathsf{RingLPN}$ *assumption implies* $[n, m-n, \mu, t - m \cdot \mathsf{poly}(n), \epsilon + 2^{-\Omega(n)}]_{\mathcal{R}_q}$-$\mathsf{RingLPN}^{s \sim \chi}$ *assumption.*

Proof. Let $\mathcal{A}$ be a distinguisher for the $\mathsf{RingLPN}^{s \sim \chi}$ problem. We construct an algorithm $\mathcal{A}'$ that uses $\mathcal{A}$ to break the $\mathsf{RingLPN}$ problem with the same advantage. Algorithm $\mathcal{A}'$ proceeds as follows:

Algorithm $\mathcal{A}'$ takes a $\mathsf{RingLPN}$ instance $(\mathbf{a}, \mathbf{b})$ as input and needs to decide whether this input is sampled from $(\mathbf{a}, \mathbf{a} \cdot s + \mathbf{e})$ for $s \leftarrow\$ \mathcal{R}_q$ or from $(\mathbf{a}, \mathbf{u})$ where $\mathbf{a}$ is uniform in $\mathcal{R}_q^m$ and $\mathbf{u}$ is uniform in $\mathcal{R}_q^m$. Algorithm $\mathcal{A}'$ translates $(\mathbf{a}, \mathbf{b})$ into an $\mathsf{RingLPN}^{s \sim \chi}$ instance $(\mathbf{a}', \mathbf{b}')$ and then runs $\mathcal{A}$ on $(\mathbf{a}', \mathbf{b}')$. Algorithm $\mathcal{A}'$ takes $(\mathbf{a}, \mathbf{b})$ as input and works as follows:

1. Parse $\mathbf{a} = (a_1, \ldots, a_{m+1})$, $\mathbf{b} = (b_1, \ldots, b_{m+1})$.
2. If $a_i \notin \mathcal{R}_q^\times$ for all $i \in [n]$, abort. Otherwise, let j be the first index in $[n]$ such that $a_j \in \mathcal{R}_q^\times$.
3. Set $a'_i \leftarrow -a_j^{-1} \cdot a_i \in \mathcal{R}_q$, $b'_i \leftarrow b_i + b_j \cdot a'_i \in \mathcal{R}_q$ for $i = n+1, \ldots, m$.
4. Set $\mathbf{a}' \leftarrow (a'_{n+1}, \ldots, a'_m) \in \mathcal{R}_q^{m-n}$, $\mathbf{b}' \leftarrow (b'_{n+1}, \ldots, b'_m) \in \mathcal{R}_q^{m-n}$.
5. Run $\mathcal{A}$ on input $(\mathbf{a}', \mathbf{b}')$ and output whatever $\mathcal{A}$ outputs.

We condition on $\mathcal{A}'$ not aborting, i.e., on the event that there exists some $i \in [n]$ such that $a_i \in \mathcal{R}_q^\times$. In our setting, since $q \equiv 1 \pmod{2n}$, we have $\mathcal{R}_q \cong \mathbb{F}_q^n$. Hence

$$\Pr[a_1 \notin \mathcal{R}_q^\times] = 1 - \left(1 - \frac{1}{q}\right)^n \leq \frac{n}{q} \leq \frac{1}{2}.$$

Therefore, the probability that $\mathcal{A}'$ aborts is

$$\Pr[\forall i \in [n],\, a_i \notin \mathcal{R}_q^\times] \leq \left(\frac{n}{q}\right)^n \leq 2^{-n}.$$

It remains to show that $(\mathbf{a}', \mathbf{b}')$ is properly distributed as a $\mathsf{RingLPN}^{s\sim\chi}$ problem instance. First, since multiplication by a unit is a permutation over $\mathcal{R}_q$, each a_i' is uniform in $\mathcal{R}_q$. Second, if the input $\mathbf{b}$ is uniform, then clearly $\mathbf{b}'$ is uniform. Third, If the input $\mathbf{b}$ satisfies $\mathbf{b} = \mathbf{a} \cdot s + \mathbf{e}$, we have

$$\begin{aligned} b_i' &= b_i + b_1 \cdot a_i' \\ &= a_i \cdot s + e_i + a_1 \cdot s + e_1 \cdot (-a_1^{-1} \cdot a_i) \\ &= a_i' \cdot e_1 + e_i \end{aligned}$$

Therefore $(\mathbf{a}', \mathbf{b}')$ is a proper $\mathsf{RingLPN}^{s\sim\chi}$ sample where the secret is $s' = e_1$. It follows that $\mathcal{A}'$ decides $\mathsf{RingLPN}^{s\sim\chi}$ with the same advantage as $\mathcal{A}$ decides $\mathsf{RingLPN}$. □

A.5 Proof of Lemma 10

Lemma 23 (Restating Lemma 10)**.** *It holds that*

$$\mathsf{SD}(\mathsf{S\text{-}Ber}(\mathbf{u}), e) \leq 2/q, \tag{16}$$

where $\mathbf{u} \leftarrow \mathbb{F}_q^2$, $e \leftarrow \mathsf{Ber}_\mu(\mathbb{F}_q)$. *In particular, if we set* $q = \lambda^{\omega(1)}$, *then*

$$\mathsf{SD}(\mathsf{S\text{-}Ber}(\mathbf{u}), \mathsf{Ber}_\mu) \leq \mathsf{negl}(\lambda) \tag{17}$$

Proof. Let D denote the distribution $\mathsf{S\text{-}Ber}(\mathbf{u})$ where $\mathbf{u} \leftarrow \mathbb{Z}_q^2$ uniformly.

Define $A = \{x_1 \in \mathbb{Z}_q : x_1 \bmod 2^\ell = 0\}$. Then $|A| = \lfloor q/2^\ell \rfloor$, so

$$\Pr[x_1 \in A] = \mu' = \frac{\lfloor q/2^\ell \rfloor}{q}$$

Let us now compute the statistical distance $\mathsf{SD}(D, e)$, where $e \leftarrow \mathsf{Ber}_\mu$, defined as:

$$\mathsf{Ber}_\mu = \begin{cases} 0 & \text{with probability } 1-\mu, \\ a \leftarrow \mathbb{Z}_q \setminus \{0\} & \text{with probability } \mu. \end{cases}$$

We analyze the pointwise difference between the distributions:

- For $a = 0$:

$$|\Pr[D = 0] - \Pr[e = 0]| = \left|(1-\mu') + \mu' \cdot \frac{1}{q} - (1-\mu)\right| \leq |\mu - \mu'| + \mu' \cdot \frac{1}{q} \leq \frac{2}{q}.$$

- For $a \neq 0$:

$$\Pr[D = a] = \Pr[x_1 \in A] \cdot \Pr[x_2 = a] = \mu' \cdot \frac{1}{q},$$
$$\Pr[e = a] = \frac{\mu}{q-1},$$
$$|\Pr[D = a] - \Pr[e = a]| \leq \left|\frac{\mu'}{q} - \frac{\mu}{q}\right| + \left|\frac{\mu}{q} - \frac{\mu}{q-1}\right| \leq \frac{1}{q^2} + \frac{\mu}{q(q-1)} \leq \frac{2}{q^2}.$$

  There are at most $q-1$ such $a \neq 0$, so:

$$\sum_{a \neq 0} |\Pr[D = a] - \Pr[e = a]| \leq (q-1) \cdot \frac{2}{q^2} \leq \frac{2}{q}.$$

Therefore, the total statistical distance is bounded by:

$$\mathsf{SD}(D, e) = \frac{1}{2} \sum_{a \in \mathbb{Z}_q} |\Pr[D = a] - \Pr[e = a]| \leq \frac{1}{2}\left(\frac{2}{q} + \frac{2}{q}\right) \leq \frac{2}{q}.$$

□

A.6 Proof of Lemma 13

Lemma 24 (restating Lemma 13). *If* LPN *is quasi-polynomially hard (Assumption 3), then* rsLPN *is also quasi-polynomially hard (Definition 26).*

Proof. Fix $i \in \{1, 2, \ldots, \log\log\log n\}$. By the reduction in Lemma 12, the reduced LPN instance has dimension

$$n_i' \;=\; d_i \cdot \log(n/d_i).$$

Noise rate regime. We first check the noise regime. The reduction preserves the noise rate, so we require the LPN hardness to hold at $\mu(n_i') = \Theta\big(n^{-1/(\log\log n)^{4i+2}}\big)$. It suffices that

$$\begin{aligned}
&\mu(n_i') = \Theta\Big(n^{-1/(\log\log n)^{4i+2}}\Big) \;=\; \Omega\Big((n_i')^{-1/\log\log n_i'}\Big) \\
\Longleftarrow\quad & \frac{\log n}{(\log\log n)^{4i+2}} \;=\; o\Big(\frac{\log n_i'}{\log\log n_i'}\Big) \\
\Longleftrightarrow\quad & \frac{\log n}{(\log\log n)^{4i+2}} \;=\; o\Big(\frac{\log(d_i \log(n/d_i))}{\log\log(d_i \log(n/d_i))}\Big) \\
\Longleftarrow\quad & \frac{\log n}{(\log\log n)^{4i+2}} \;=\; o\Big(\frac{\log d_i}{\log\log d_i}\Big),
\end{aligned} \tag{18}$$

where the last implication uses that $x \mapsto \log x / \log\log x$ is increasing. Plugging $d_i = \Theta\big(n^{1/(\log\log n)^{4i}}\big)$

yields

$$\begin{aligned} \iff \frac{\log n}{(\log\log n)^{4i+1}} &= o\Big(\frac{\log n^{1/(\log\log n)^{4i}}}{\log\log n^{1/(\log\log n)^{4i}}}\Big) \\ \iff \frac{\log n}{(\log\log n)^{4i+2}} &= o\Big(\frac{\log n}{(\log\log n)^{4i}} \cdot \frac{1}{\log\log n - (4i)\log\log\log n}\Big). \end{aligned} \tag{19}$$

Since $(4i) \le O(\log\log\log n)$, we have $(4i)\log\log\log n = O\big((\log\log\log n)^2\big) = o(\log\log n)$. Hence

$$\begin{aligned} \iff (\log\log n)^2 &= \omega\big(\log\log n - 4i\log\log\log n\big) \\ \iff (\log\log n)^2 &= \omega(\log\log n), \end{aligned} \tag{20}$$

as required, and the required noise regime for LPN is met.

Time/sample/advantage parameters. It remains to check that the running time, number of samples, and distinguishing advantage fall within the assumed LPN hardness window at dimension n_i'. It is enough to show that, for any constant $c > 0$,

$$\begin{aligned} 2^{c\log^{1.01} n} &= 2^{O(\log^{1.1} n_i')} \\ \impliedby \log^{1.01} n &= o\big(\log^{1.1} n_i'\big) \\ \impliedby \log^{1.01} n &= o\Big(\big(\log(d_i \log(n/d_i))\big)^{1.1}\Big) \\ \impliedby \log^{1.01} n &= o\big((\log d_i)^{1.1}\big). \end{aligned} \tag{21}$$

Substituting $d_i = \Theta\big(n^{1/(\log\log n)^{4i}}\big)$ gives

$$\begin{aligned} \iff \log^{1.01} n &= o\Big(\big(\log n^{1/(\log\log n)^{4i}}\big)^{1.1}\Big) \\ \iff \log^{1.01} n &= o\Big(\frac{\log^{1.1} n}{(\log\log n)^{1.1(4i)}}\Big), \end{aligned} \tag{22}$$

which holds because $(\log\log n)^{1.1(4i)} = O(\log\log n)^{\log\log\log n} = o(\log^{0.01} n)$. □

References

[ABW10] Benny Applebaum, Boaz Barak, and Avi Wigderson. Public-key cryptography from different assumptions. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC*, pages 171–180, 2010.

[ADI+17] Benny Applebaum, Ivan Damgård, Yuval Ishai, Michael Nielsen, and Lior Zichron. Secure arithmetic computation with constant computational overhead. In *Advances in Cryptology - CRYPTO*, volume 10401, pages 223–254, 2017.

[AIK08] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. On pseudorandom generators with linear stretch in NC^0. *Comput. Complex.*, 17(1):38–69, 2008.

[Ale03] Michael Alekhnovich. More on average case vs approximation complexity. In *44th Symposium on Foundations of Computer Science, FOCS*, pages 298–307, 2003.

[Ale11] Michael Alekhnovich. More on average case vs approximation complexity. *Comput. Complex.*, 20(4):755–786, 2011.

[AMR25] Damiano Abram, Giulio Malavolta, and Lawrence Roy. Slightly sublinear trapdoor hash functions and PIR from low-noise LPN. In *Theory of Cryptography - 23rd International Conference, TCC (to appear)*, 2025.

[AN25] Benny Applebaum and Oded Nir. The meta-complexity of secret sharing. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing, STOC*, pages 965–976. ACM, 2025.

[App14] Benny Applebaum. Bootstrapping obfuscators via fast pseudorandom functions. In *Advances in Cryptology - ASIACRYPT*, pages 162–172, 2014.

[App16] Benny Applebaum. Cryptographic hardness of random local functions - survey. *Comput. Complex.*, 25(3):667–722, 2016.

[AR16] Benny Applebaum and Pavel Raykov. Fast pseudorandom functions based on expander graphs. In *Theory of Cryptography - 14th International Conference, TCC*, volume 9985, pages 27–56, 2016.

[BBTV25] Kiril Bangachev, Guy Bresler, Stefan Tiegel, and Vinod Vaikuntanathan. Near-optimal time-sparsity trade-offs for solving noisy linear equations. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing, STOC*, pages 1910–1920, 2025.

[BCG+20] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators from Ring-LPN. In *Advances in Cryptology - CRYPTO*, volume 12171, pages 387–416, 2020.

[BCH86] Paul Beame, Stephen A. Cook, and H. James Hoover. Log depth circuits for division and related problems. *SIAM J. Comput.*, 15(4):994–1003, 1986.

[BDGJ20] Colin Boyd, Gareth T. Davies, Kristian Gjøsteen, and Yao Jiang. Fast and secure updatable encryption. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part I*, volume 12170 of *Lecture Notes in Computer Science*, pages 464–493, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Cham, Switzerland.

[BEKS20] Dan Boneh, Saba Eskandarian, Sam Kim, and Maurice Shih. Improving speed and security in updatable encryption schemes. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020, Part III*, volume 12493 of *Lecture Notes in Computer Science*, pages 559–589, Daejeon, South Korea, December 7–11, 2020. Springer, Cham, Switzerland.

[BFKL93] Avrim Blum, Merrick L. Furst, Michael J. Kearns, and Richard J. Lipton. Cryptographic primitives based on hard learning problems. In Douglas R. Stinson, editor, *Advances in Cryptology - CRYPTO*, volume 773, pages 278–291, 1993.

[BHKN13] Itay Berman, Iftach Haitner, Ilan Komargodski, and Moni Naor. Hardness preserving reductions via cuckoo hashing. In *Theory of Cryptography - 10th Theory of Cryptography Conference, TCC*, volume 7785, pages 40–59, 2013.

[BHKN19] Itay Berman, Iftach Haitner, Ilan Komargodski, and Moni Naor. Hardness-preserving reductions via Cuckoo hashing. *J. Cryptol.*, 32(2):361–392, 2019.

[BIP+18] Dan Boneh, Yuval Ishai, Alain Passelègue, Amit Sahai, and David J. Wu. Exploring crypto dark matter: New simple PRF candidates and their applications. In *Theory of Cryptography - 16th International Conference, TCC*, volume 11240, pages 699–729, 2018.

[BJKL21] Fabrice Benhamouda, Aayush Jain, Ilan Komargodski, and Huijia Lin. Multiparty reusable non-interactive secure computation from LWE. In *Advances in Cryptology - EUROCRYPT*, volume 12697, pages 724–753, 2021.

[BJMM12] Anja Becker, Antoine Joux, Alexander May, and Alexander Meurer. Decoding random binary linear codes in $2^{n/20}$: How $1 + 1 = 0$ improves information set decoding. In *Advances in Cryptology - EUROCRYPT*, volume 7237, pages 520–536, 2012.

[BKW03] Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *J. ACM*, 50(4):506–519, 2003.

[BL12] Daniel J. Bernstein and Tanja Lange. Never trust a bunny. In *Radio Frequency Identification. Security and Privacy Issues - 8th International Workshop, RFIDSec*, volume 7739, pages 137–148, 2012.

[BLMR13] Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. Key homomorphic prfs and their applications. In *Advances in Cryptology - CRYPTO*, volume 8042, pages 410–428, 2013.

[BLP11] Daniel J. Bernstein, Tanja Lange, and Christiane Peters. Smaller decoding exponents: Ball-collision decoding. In *Advances in Cryptology - CRYPTO*, volume 6841, pages 743–760, 2011.

[BLRS26] Jaroslaw Blasiok, Paul Lou, Alon Rosen, and Madhu Sudan. Expanders meet reed-muller: Easy instances of noisy k-xor. *CoRR*, abs/2604.04188, 2026.

[BLVW19] Zvika Brakerski, Vadim Lyubashevsky, Vinod Vaikuntanathan, and Daniel Wichs. Worst-case hardness for LPN and cryptographic hashing via code smoothing. In *Advances in Cryptology - EUROCRYPT*, volume 11478, pages 619–635, 2019.

[BMM24] Amos Beimel, Tal Malkin, and Noam Mazor. Structural lower bounds on black-box constructions of pseudorandom functions. In *Advances in Cryptology - CRYPTO*, volume 14924, pages 459–488, 2024.

[BP14] Abhishek Banerjee and Chris Peikert. New and improved key-homomorphic pseudorandom functions. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 353–370, Santa Barbara, CA, USA, August 17–21, 2014. Springer Berlin Heidelberg, Germany.

[BPR12] Abhishek Banerjee, Chris Peikert, and Alon Rosen. Pseudorandom functions and lattices. In *Advances in Cryptology - EUROCRYPT*, volume 7237, pages 719–737, 2012.

[BQ12] Andrej Bogdanov and Youming Qiao. On the security of goldreich's one-way function. *Comput. Complex.*, 21(1):83–127, 2012.

[BR17] Andrej Bogdanov and Alon Rosen. Pseudorandom functions: Three decades later. In *Tutorials on the Foundations of Cryptography*, pages 79–158. Springer International Publishing, 2017.

[CC98] Anne Canteaut and Florent Chabaud. A new algorithm for finding minimum-weight words in a linear code: Application to McEliece's cryptosystem and to narrow-sense BCH codes of length 511. *IEEE Trans. Inf. Theory*, 44(1):367–378, 1998.

[CHKV25] Henry Corrigan-Gibbs, Alexandra Henzinger, Yael Tauman Kalai, and Vinod Vaikuntanathan. Somewhat homomorphic encryption from linear homomorphism and sparse LPN. In *Advances in Cryptology - EUROCRYPT*, volume 15602, pages 3–33, 2025.

[CM21] Geoffroy Couteau and Pierre Meyer. Breaking the circuit size barrier for secure computation under quasi-polynomial LPN. In *Advances in Cryptology - EUROCRYPT*, volume 12697, pages 842–870, 2021.

[CSZ25] Xue Chen, Wenxuan Shu, and Zhaienhe Zhou. Algorithms for sparse LPN and LSPN against low-noise (extended abstract). In *The Thirty Eighth Annual Conference on Learning Theory*, volume 291, pages 1091–1093, 2025.

[DI06] Ivan Damgård and Yuval Ishai. Scalable secure multiparty computation. In Cynthia Dwork, editor, *Advances in Cryptology - CRYPTO*, volume 4117 of *Lecture Notes in Computer Science*, pages 501–520, 2006.

[DIJL23] Quang Dao, Yuval Ishai, Aayush Jain, and Huijia Lin. Multi-party homomorphic secret sharing and sublinear MPC from sparse LPN. In *Advances in Cryptology - CRYPTO*, volume 14082, pages 315–348, 2023.

[DJK25] Youlong Ding, Aayush Jain, and Ilan Komargodski. A new approach for LPN-based pseudorandom functions: Low-depth and key-homomorphic. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing, STOC*, pages 1898–1909, 2025.

[DP12] Ivan Damgård and Sunoo Park. How practical is public-key encryption based on LPN and ring-LPN? Cryptology ePrint Archive, Paper 2012/699, 2012.

[DS15] Nico Döttling and Dominique Schröder. Efficient pseudorandom functions via on-the-fly adaptation. In *Advances in Cryptology - CRYPTO*, volume 9215, pages 329–350, 2015.

[EKM17] Andre Esser, Robert Kübler, and Alexander May. LPN decoded. In *Advances in Cryptology - CRYPTO*, volume 10402, pages 486–514, 2017.

[EPRS17] Adam Everspaugh, Kenneth G. Paterson, Thomas Ristenpart, and Samuel Scott. Key rotation for authenticated encryption. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part III*, volume 10403 of *Lecture Notes in Computer Science*, pages 98–129, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Cham, Switzerland.

[FLY22] Zhiyuan Fan, Jiatu Li, and Tianqi Yang. The exact complexity of pseudorandom functions and the black-box natural proof barrier for bootstrapping results in computational complexity. In *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 962–975, 2022.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.

[GHJS25] Riddhi Ghosal, Isaac M. Hair, Aayush Jain, and Amit Sahai. Using the planted clique conjecture for cryptography: Public-key encryption from planted clique and noisy k-LIN over expanders. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing, STOC*, pages 1921–1932, 2025.

[GJL15] Qian Guo, Thomas Johansson, and Carl Löndahl. A new algorithm for solving ring-lpn with a reducible polynomial. *IEEE Trans. Inf. Theory*, 61(11):6204–6212, 2015.

[Gol00] Oded Goldreich. Candidate one-way functions based on expander graphs. Cryptology ePrint Archive, Report 2000/063, 2000. https://eprint.iacr.org/2000/063.

[GUV09] Venkatesan Guruswami, Christopher Umans, and Salil P. Vadhan. Unbalanced expanders and randomness extractors from Parvaresh-Vardy codes. *J. ACM*, 56(4):20:1–20:34, 2009.

[GVW12] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption with bounded collusions via multi-party computation. In *Advances in Cryptology - CRYPTO*, volume 7417, pages 162–179. Springer, 2012.

[HAB02] William Hesse, Eric Allender, and David A. Mix Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *J. Comput. Syst. Sci.*, 65(4):695–716, 2002.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28(4):1364–1396, 1999.

[HKL+12] Stefan Heyse, Eike Kiltz, Vadim Lyubashevsky, Christof Paar, and Krzysztof Pietrzak. Lapin: An efficient authentication protocol based on ring-LPN. In *Fast Software Encryption - 19th International Workshop, FSE*, volume 7549, pages 346–365, 2012.

[HRV13] Iftach Haitner, Omer Reingold, and Salil P. Vadhan. Efficiency improvements in constructing pseudorandom generators from one-way functions. *SIAM J. Comput.*, 42(3):1405–1430, 2013.

[IKOS08] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Cryptography with constant computational overhead. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing*, pages 433–442. ACM, 2008.

[IL90] Russell Impagliazzo and Leonid A. Levin. No better ways to generate hard NP instances than picking uniformly at random. In *31st Annual Symposium on Foundations of Computer Science*, pages 812–821, St. Louis, MO, USA, October 22–24, 1990. IEEE Computer Society Press.

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 60–73, 2021.

[JLS22] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from LPN over $\mathbb{F}_p$, DLIN, and PRGs in NC^0. In *Advances in Cryptology - EUROCRYPT*, volume 13275, pages 670–699, 2022.

[JLS24] Aayush Jain, Huijia Lin, and Sagnik Saha. A systematic study of sparse LWE. In *Advances in Cryptology - CRYPTO*, volume 14922, pages 210–245, 2024.

[Kim20] Sam Kim. Key-homomorphic pseudorandom functions from LWE with small modulus. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part II*, volume 12106 of *Lecture Notes in Computer Science*, pages 576–607, Zagreb, Croatia, May 10–14, 2020. Springer, Cham, Switzerland.

[Kir11] Paul Kirchner. Improved generalized birthday attack. *IACR Cryptol. ePrint Arch.*, page 377, 2011.

[KLR19] Michael Klooß, Anja Lehmann, and Andy Rupp. (R)CCA secure updatable encryption with integrity protection. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part I*, volume 11476 of *Lecture Notes in Computer Science*, pages 68–99, Darmstadt, Germany, May 19–23, 2019. Springer, Cham, Switzerland.

[KT22] Itay Kalev and Amnon Ta-Shma. Unbalanced expanders from multiplicity codes. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM*, volume 245, pages 12:1–12:14, 2022.

[KV89] Michael J. Kearns and Leslie G. Valiant. Cryptographic limitations on learning Boolean formulae and finite automata. In *21st Annual ACM Symposium on Theory of Computing*, pages 433–444, Seattle, WA, USA, May 15–17, 1989. ACM Press.

[Lev85] Leonid A. Levin. One-way functions and pseudorandom generators. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing*, pages 363–365, 1985.

[Lev87] Leonid A. Levin. One-way functions and pseudorandom generators. *Comb.*, 7(4):357–363, 1987.

[LF06] Éric Levieil and Pierre-Alain Fouque. An improved LPN algorithm. In *Security and Cryptography for Networks, 5th International Conference, SCN*, volume 4116 of *Lecture Notes in Computer Science*, pages 348–359, 2006.

[LLPT23] Hanjun Li, Huijia Lin, Antigoni Polychroniadou, and Stefano Tessaro. LERNA: Secure single-server aggregation via key-homomorphic masking. In Jian Guo and Ron Steinfeld, editors, *Advances in Cryptology – ASIACRYPT 2023, Part I*, volume 14438 of *Lecture Notes in Computer Science*, pages 302–334, Guangzhou, China, December 4–8, 2023. Springer, Singapore, Singapore.

[LP15] Helger Lipmaa and Kateryna Pavlyk. Analysis and implementation of an efficient ring-lpn based commitment scheme. In *Cryptology and Network Security - 14th International Conference, CANS*, volume 9476, pages 160–175, 2015.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *Advances in Cryptology - EUROCRYPT*, volume 6110, pages 1–23, 2010.

[LPR13a] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. *J. ACM*, 60(6):43:1–43:35, 2013.

[LPR13b] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. A toolkit for ring-lwe cryptography. In *Advances in Cryptology - EUROCRYPT*, pages 35–54, 2013.

[LT18] Anja Lehmann and Björn Tackmann. Updatable encryption with post-compromise security. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part III*, volume 10822 of *Lecture Notes in Computer Science*, pages 685–716, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Cham, Switzerland.

[LW09] Allison B. Lewko and Brent Waters. Efficient pseudorandom functions from the decisional linear assumption and weaker variants. In *Proceedings of the 2009 ACM Conference on Computer and Communications Security, CCS*, pages 112–120, 2009.

[LWYY24] Hanlin Liu, Xiao Wang, Kang Yang, and Yu Yu. The hardness of LPN over any integer ring and field for PCG applications. In *Advances in Cryptology - EUROCRYPT*, volume 14656, pages 149–179, 2024.

[Lyu05] Vadim Lyubashevsky. The parity problem in the presence of noise, decoding random linear codes, and the subset sum problem. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, pages 378–389, 2005.

[McE78] R. J. McEliece. A Public-Key Cryptosystem Based On Algebraic Coding Theory. *Deep Space Network Progress Report*, 44:114–116, January 1978.

[MMT11] Alexander May, Alexander Meurer, and Enrico Thomae. Decoding random linear codes in $\tilde{O}(2^{0.054n})$. In *Advances in Cryptology - ASIACRYPT*, volume 7073, pages 107–124, 2011.

[MV12] Eric Miles and Emanuele Viola. Substitution-permutation networks, pseudorandom functions, and natural proofs. In *Advances in Cryptology - CRYPTO*, volume 7417, pages 68–85, 2012.

[MV15] Eric Miles and Emanuele Viola. Substitution-permutation networks, pseudorandom functions, and natural proofs. *J. ACM*, 62(6):46:1–46:29, 2015.

[NPR99] Moni Naor, Benny Pinkas, and Omer Reingold. Distributed pseudo-random functions and KDCs. In Jacques Stern, editor, *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 327–346, Prague, Czech Republic, May 2–6, 1999. Springer Berlin Heidelberg, Germany.

[NR95] Moni Naor and Omer Reingold. Synthesizers and their application to the parallel construction of pseudo-random functions. In *36th Annual Symposium on Foundations of Computer Science, FOCS*, pages 170–181. IEEE Computer Society, 1995.

[NR97] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *38th Annual Symposium on Foundations of Computer Science, FOCS*, pages 458–467, 1997.

[NR99] Moni Naor and Omer Reingold. Synthesizers and their application to the parallel construction of pseudo-random functions. *J. Comput. Syst. Sci.*, 58(2):336–375, 1999.

[NR04] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. *J. ACM*, 51(2):231–262, 2004.

[NRR02] Moni Naor, Omer Reingold, and Alon Rosen. Pseudorandom functions and factoring. *SIAM J. Comput.*, 31(5):1383–1404, 2002.

[OST19] Igor Carboni Oliveira, Rahul Santhanam, and Roei Tell. Expander-based cryptography meets natural proofs. In *10th Innovations in Theoretical Computer Science Conference, ITCS*, volume 124, pages 18:1–18:14, 2019.

[OST22] Igor C. Oliveira, Rahul Santhanam, and Roei Tell. Expander-based cryptography meets natural proofs. *Comput. Complex.*, 31(1):4, 2022.

[OW14] Ryan O'Donnell and David Witmer. Goldreich's PRG: evidence for near-optimal polynomial stretch. In *IEEE 29th Conference on Computational Complexity, CCC*, pages 1–12. IEEE Computer Society, 2014.

[Pra62] Eugene Prange. The use of information sets in decoding cyclic codes. *IRE Trans. Inf. Theory*, 8(5):5–9, 1962.

[RR97] Alexander A. Razborov and Steven Rudich. Natural proofs. *J. Comput. Syst. Sci.*, 55(1):24–35, 1997.

[RVV24] Seyoon Ragavan, Neekon Vafa, and Vinod Vaikuntanathan. Indistinguishability obfuscation from bilinear maps and LPN variants. In *Theory of Cryptography - 22nd International Conference, TCC*, volume 15367, pages 3–36, 2024.

[Sch18] Peter Scholl. Extending oblivious transfer with low communication via key-homomorphic PRFs. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018: 21st International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 10769 of *Lecture Notes in Computer Science*, pages 554–583, Rio de Janeiro, Brazil, March 25–29, 2018. Springer, Cham, Switzerland.

[Ste89] Jacques Stern. A method for finding codewords of small weight. In *Coding Theory and Applications*, pages 106–113, 1989.

[Val84] Leslie G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, 1984.

[Vio15] Emanuele Viola. The communication complexity of addition. *Comb.*, 35(6):703–747, 2015.

[VZ12] Salil P. Vadhan and Colin Jia Zheng. Characterizing pseudoentropy and simplifying pseudorandom generator constructions. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC*, pages 817–836. ACM, 2012.

[YZ16] Yu Yu and Jiang Zhang. Cryptography with auxiliary input and trapdoor from constant-noise LPN. In *Advances in Cryptology - CRYPTO*, volume 9814, pages 214–243, 2016.

[YZW+19] Yu Yu, Jiang Zhang, Jian Weng, Chun Guo, and Xiangxue Li. Collision resistant hashing from sub-exponential learning parity with noise. In *Advances in Cryptology - ASIACRYPT*, volume 11922, pages 3–24, 2019.

[Zha25] Mark Zhandry. Optimal traitor tracing from pairings. In *Advances in Cryptology - EUROCRYPT*, volume 15603, pages 305–335, 2025.