# Tree Encodings I: How to Authenticate a Truth Table

Damiano Abram* Giulio Malavolta† Lawrence Roy‡

**Abstract**

We propose a new method to construct homomorphic authentication codes for *truth tables* of read-once matrix branching programs, extending the celebrated homomorphic lattice encodings [Boneh et al., Eurocrypt 2014]. Our approach relies on the hardness of the decomposed learning with errors problem (DLWE), a recently introduced modification of Regev's LWE assumption.

We then use this new technical tool to make progress on several open problems in the literature. Specifically, we obtain:

- A constrained pseudorandom function (PRF), where the evaluation of the PRF on the master key does not depend on the complexity of the constraint, except for its circuit depth.
- A way to securely compress and re-expand LWE samples in the plain model.
- An adaptively secure broadcast encryption scheme, with ciphertext and secret keys growing poly-logarithmically with the size of the encrypted set.
- A pseudorandom obfuscation for all puncturable PRFs, additionally assuming the existence of sub-exponentially secure indistinguishability obfuscation (iO).

None of the above mentioned primitives was known to exist from lattice assumptions. As a bonus result, we also obtain a conceptually simple and direct heuristic construction of iO based on lattice techniques, which is not based on the functional encryption-to-iO paradigm. We provide evidence that this approach can be used to build provably secure obfuscation for simple functionalities such as sampling lattice preimages using a hidden trapdoor.

*University of Edinburgh
†Bocconi University
‡Aarhus University, IBM Research Zürich

# Contents


**1 Introduction** **2**
- 1.1 Our Results . . . 3
- 1.2 Concurrent and Subsequent Work . . . 6
- 1.3 Technical Outline: Tree Encodings . . . 6
- 1.4 Technical Outline: Applications . . . 11

**2 Preliminaries** **16**
- 2.1 Linear Algebra . . . 16
- 2.2 Matrix Branching Programs . . . 17
- 2.3 Discrete Gaussians and Lattices . . . 18
- 2.4 Homomorphic Encodings . . . 20

**3 Tree Encodings** **21**
- 3.1 Hashing Branching Programs . . . 22
- 3.2 Homomorphic Operations . . . 23
- 3.3 Flattening . . . 25
- 3.4 Putting Things Together . . . 27

**4 Laconic Shift-Hiding Pseudorandom Function** **30**
- 4.1 Laconic Function Evaluation . . . 30
- 4.2 Our Scheme . . . 32
- 4.3 Removing the Input Dependence . . . 35
- 4.4 A Heuristic Approach to Obfuscation . . . 37

**5 Compressing LWE Samples** **37**
- 5.1 The Compression Algorithm . . . 39
- 5.2 Derandomized Indistinguishability . . . 41

**6 Broadcast Encryption** **46**
- 6.1 Definition . . . 46
- 6.2 Our Scheme . . . 47

**7 Pseudorandom Obfuscation** **50**


# 1 Introduction

Lattice-based cryptography [Ajt96, Reg05] has seen a remarkable success over the past decade, in part due to its conjectured *post-quantum* security, in part due to the vast number of new cryptographic primitives enabled by lattices. A central property for lattice-based cryptography is the ability of *computing on encrypted data*, which is the cornerstone of primitives such as fully-homomorphic encryption [Gen09, BV11], attribute-based encryption [GVW13, BGG+14], laconic function evaluation [QWW18], traitor tracing [GKW18], or succinct arguments [CJJ22], to mention a few. A central technical tool in this area is the notion of *lattice encodings*, introduced in

[BGG+14]. An encoding of a message $\mathbf{x}$ with respect to a matrix $\mathbf{M}$ is defined to be

$$\mathbf{c}^\intercal \approx \mathbf{s}^\intercal \mathbf{G}(\mathbf{M} - \mathbf{x}^\intercal \otimes \mathbf{I}_n)$$

where $\mathbf{G}$ is the gadget matrix [MP12]. The power of such encodings comes from their *homomorphic properties*: For any function $f$ computable by a circuit of depth smaller than a parameter $\delta$, one can compute low-norm matrices $\mathbf{H}_{f,\mathbf{x}}$ and $\mathbf{K}_f$ such that

$$\mathbf{G}(\mathbf{M} - \mathbf{x}^\intercal \otimes \mathbf{I}_n) \cdot \mathbf{H}_{f,\mathbf{x}} = \mathbf{G}(\mathbf{M}\mathbf{K}_f - f(\mathbf{x})^\intercal \otimes \mathbf{I}_n). \tag{1}$$

This allows one to publicly evaluate $f$ on the encoded input $\mathbf{x}$, by computing

$$\mathbf{c}^\intercal \cdot \mathbf{H}_{f,\mathbf{x}} \approx \mathbf{s}^\intercal \mathbf{G}(\mathbf{M}\mathbf{K}_f - f(\mathbf{x})^\intercal \otimes \mathbf{I}_n).$$

A recent line of work [Wee24, Wee25, AMR25b, AMR25a] shows how to remove the dependency of the encoding on the size of $\mathbf{x}$, enabling a series of new cryptographic applications. In a nutshell, this leads to a new form of *compressed encoding*, consisting of

$$\mathbf{c}^\intercal \approx \mathbf{s}^\intercal \mathbf{G} \cdot (\mathbf{M} - \mathsf{Hash}(\mathbf{x})^\intercal \otimes \mathbf{I}_n), \qquad \mathbf{z}^\intercal \approx \mathbf{s}^\intercal \mathbf{Z},$$

where $\mathsf{Hash}$ is a deterministic and *linear* hash function and $\mathbf{Z}$ is a public matrix, independent of $\mathbf{x}$. The two components can be publicly *expanded* back to a regular uncompressed [BGG+14] encoding, using the following expansion equation:

$$\mathbf{G}(\mathbf{M} - \mathsf{Hash}(\mathbf{x})^\intercal \otimes \mathbf{I}_n) \cdot \mathbf{P} + \mathbf{Z} \cdot \mathbf{Q}_\mathbf{x} = \mathbf{G}(\mathbf{M}\mathbf{P} - \mathbf{x}^\intercal \otimes \mathbf{I}_n) \tag{2}$$

for some low-norm matrices $\mathbf{P}$ and $\mathbf{Q}_\mathbf{x}$. Then, it is not hard to see how one can combine Equation (1) and Equation (2) to support homomorphic evaluation on compressed encodings:

$$(\mathbf{G}(\mathbf{M} - \mathsf{Hash}(\mathbf{x})^\intercal \otimes \mathbf{I}_n) \cdot \mathbf{P} + \mathbf{Z} \cdot \mathbf{Q}_\mathbf{x})\mathbf{H}_{f,\mathbf{x}} = \mathbf{G}(\mathbf{M}\mathbf{P} - \mathbf{x}^\intercal \otimes \mathbf{I}_n)\mathbf{H}_{f,\mathbf{x}} = \mathbf{G}(\mathbf{M}\mathbf{P}\mathbf{K}_f - f(\mathbf{x})^\intercal \otimes \mathbf{I}_n).$$

This fact was also recently used in [Wee25] in the context of functional commitments [IKO07, BC12, LRY16]. For applications, it is important to minimize the computation needed to derive the matrix $\mathbf{M}\mathbf{P}\mathbf{K}_f$, which is needed to verify the opening of the functional commitment (we informally refer to this object as the *opening matrix*). Motivated by this problem, [Wee25] introduced a new variant of the encodings, where the opening matrix can be decomposed as the product of a $f$-dependent matrix $\mathbf{M}_f$ (corresponding to the commitment to $f$) and an input-dependent matrix $\mathbf{P}_\mathbf{x}$, where the latter is computable in time sublinear in $f$. To be more precise, $\mathbf{P}_\mathbf{x}$ can be computed in time that only depends (polynomially) on the bound on the circuit depth $\delta$, whereas the depth of the computation scales quasi-linearly in $\delta$.

### 1.1 Our Results

Our main contribution is a new type of lattice encodings, which we refer to as *tree* encodings. The key property is that the opening matrix $\mathbf{M}_f \cdot \mathbf{P}_\mathbf{x}$ is such that $\mathbf{P}_\mathbf{x}$ is computable in depth *polylogarithmic* in $\delta$. The significance of this fact is that the function computing $\mathbf{P}_\mathbf{x}$ is itself now *within* the class of circuits supported by the homomorphic evaluation. This was not the case in prior constructions, such as [Wee25], since the depth-dependence would create a circularity in the choice of the parameters. As we shall discuss in a moment, this new ability of homomorphically

computing $\mathbf{P}_{\mathbf{x}}$ will be the key to unlocking a series of new applications in cryptography. At a high-level, we achieve this by constructing a linear hash function that allows us to (i) compute the hash of a *truth-table* of a read-once matrix branching program $\{\mathsf{BP}(\alpha)\}_\alpha$ in polynomial time, whereas the obvious algorithm would require time exponential in the input length. Moreover, we can (ii) homomorphically evaluate any arithmetic circuit over hash digests and, consistently, over the "leaves" $\mathsf{BP}(\alpha)$. We refer the reader to Section 1.3 for a more detailed description of the tree encodings and their associated algorithms.

All of our results rely on the hardness of the *decomposed LWE* problem [AMR25a], a recent extension of Regev's LWE problem [Reg05]. In [AMR25a] it is shown that the hardness of the decomposed LWE problem is implied by the hardness of the succinct LWE problem [Wee24]. Below we discuss our applications in more detail.

**Application I: Laconic Shift-Hiding PRFs.** A shift-hiding pseudorandom function (PRF) [PS18] is a regular PRF except that it has the additional ability to issue *shifted* keys for an arbitrary function $f$ so that evaluating the shifted PRF on an input $x$ results in the output "shifted" by $f(x)$. Furthermore, the shifted key must computationally hide $f$. Shift-hiding PRFs are known to imply constrained PRFs [BV15b, BLW17] and correlation-intractable hash functions [CGH98, LV22].

Using our tree encodings, we obtain the first *laconic* shift-hiding PRF, where the (non-shifted) PRF evaluation belongs to the class of supported shifts. In particular, the complexity of the operation is independent of the complexity of shifted function $f$, except for its circuit depth. This seemingly technical requirement turns out to be crucial for subsequent applications since it enables the shift function to recursively evaluate its own PRF, akin to bootstrapping in fully-homomorphic encryption [Gen09].

**Theorem 1** (Informal)**.** *Assuming the hardness of the decomposed LWE problem, there exists a laconic shift-hiding PRF for all functions.*

**Application II: Securely Compressing LWE Samples.** We show a method to securely *compress-and-expand* LWE samples with respect to a matrix $\mathbf{X}$. To exemplify the usefulness of this tool, suppose that Alice holds an LWE secret $\mathbf{s}$ and wants to provide an LWE sample $\mathbf{s}^\intercal\mathbf{X} + \mathbf{e}^\intercal$ to Bob. Using our compression algorithm, Alice can send $\mathbf{s}^\intercal \cdot \mathsf{Hash}(\mathbf{X}) + \mathbf{e}^\intercal$ to Bob, where the size of $\mathsf{Hash}(\mathbf{X})$ is poly-logarithmic in the number of columns of $\mathbf{X}$. Finally, Bob can locally expand the sample sent by Alice to obtain an LWE sample with respect to $\mathbf{X}$.

For security, we show that if $\mathbf{X}$ is random, Alice's message is indistinguishable from random, under the hardness of decomposed LWE. This ensures that the secrecy of $\mathbf{s}$ is preserved. This is summarized by the following theorem.

**Theorem 2** (Informal)**.** *There exist polynomial time algorithms* $(\mathsf{Hash}, \mathsf{Expand})$ *such that, if* $\mathbf{h}^\intercal \approx \mathbf{s}^\intercal \cdot \mathsf{Hash}(\mathbf{X})$*, then* $\mathsf{Expand}(\mathbf{h}, \mathbf{X}) \approx \mathbf{s}^\intercal \cdot \mathbf{X}$*. Moreover, if* $\mathbf{X}$ *is random, assuming the hardness of decomposed LWE,* $\mathbf{h}$ *is indistinguishable from random.*

We highlight that this primitive does not give an *oblivious LWE sampler* [WW21] and, indeed, we do not expect it to enable the construction of indistinguishability obfuscation. This is because, on top of compression, *oblivious LWE samplers* provide weak security guarantees even when we leak the LWE secret.

**Application III: Adaptively-Secure Broadcast Encryption.** Broadcast encryption [FN94] allows one to broadcast the same message to a set of $N$ participants. The notion requires the size of the ciphertext to be sublinear (ideally poly-logarithmic) in $N$. Traditionally, this notion entailed the presence of a trusted party that pre-distributes keys to the receivers, but recent work extended it to the *distributed* settings [BZ14] where each receiver samples its own key pair.

Arguably, the correct notion of security for broadcast encryption is *adaptive security*: The message should be hidden to the eyes of a distinguisher who can select the recipient set adaptively, after seeing the keys of the receivers. Our main result is the first lattice-based (distributed) broadcast encryption that is adaptively secure in the plain model.

**Theorem 3** (Informal)**.** *Assuming the hardness of the decomposed LWE problem, there exists an adaptively-secure (distributed, resp.) broadcast encryption scheme where the size of the private (public, resp.) keys and ciphertexts is poly-logarithmic in $N$.*

Note that we achieve the stronger notion of distributed broadcast, but our scheme is also the first lattice-based (non-distributed) broadcast encryption satisfying the above security notion in the plain model. Our main technical contribution is an *unbounded* broadcast encryption scheme with essentially optimal parameters satisfying the weaker notion of semi-static security, then we invoke a recent compiler that generically transforms it into an adaptively-secure scheme in the standard model [HWW25]. We shall mention that the compilation process from [HWW25] introduces a large unstructured common reference string (linear in $N$), which is then inherited by our main construction. We leave open the problem of reducing the size of the common reference string to poly-logarithmic in $N$, while maintaining adaptive security.

Prior constructions achieving adaptive security (both in the distributed and traditional settings) required bilinear pairings [BW06, Wat09] or witness encryption [FWW23] or were only proven secure in the random oracle model [CHW25]. For context, witness encryption candidates based on lattices exist [Tsa22, VWW22] but the validity of the underlying assumptions has recently been called into question [BÜW24, DJM+25]. We also mention some recent lattice-based schemes satisfying the weaker notion of *selective security* [WW25], where the attacker has to commit to the challenge set ahead of time, prior to seeing the keys of the honest parties.

**Application IV: Pseudorandom Obfuscation.** An obfuscation scheme allows one to make a program unintelligible, without altering its functionality. The standard notion of security for an obfuscation scheme is indistinguishability obfuscation (iO) [BGI+01], which requires that the obfuscation of two functionally-equivalent programs must be computationally indistinguishable. A recent work [BDJ+25] proposes a variant of this notion, called *pseudorandom obfuscation* for keyed functions. The security of the primitive postulates that the obfuscation *itself* should be computationally indistinguishable from uniform bits, provided that the obfuscated function is also pseudorandom. However, in the same work [BDJ+25] the authors showed strong impossibility results for pseudorandom obfuscation for general functions.

This leaves open the question on what kind of functions admit a pseudorandom obfuscator. We make progress on this question, showing that all *puncturable* pseudorandom functions admit a pseudorandom obfuscation.

**Theorem 4** (Informal)**.** *Assuming the sub-exponential hardness of the decomposed LWE problem and the existence of sub-exponentially secure iO, there exists a pseudorandom obfuscation for all puncturable functions.*

A puncturable function [SW14] allows one to "puncture out" a point in its domain so that, even given the ability of evaluating the function on all other points, the output at the punctured point is computationally indistinguishable from uniform. Conditioned on the validity of the decomposed LWE assumption, this confirms a conjecture from [BDJ+25].

**Application V: A Heuristic Obfuscation Scheme.** As a bonus result, we show how our laconic shift-hiding PRF immediately yields a fully lattice-based construction of general purpose obfuscation for any circuit $C$ as follows: Set the shift function to be $C(x) - \mathsf{Eval}(K, x)$, then the shifted evaluation algorithm returns

$$\mathsf{Eval}(K, x) + C(x) - \mathsf{Eval}(K, x) = C(x).$$

Note that the shift function is only well-defined because of the *laconic property*: The shift depends on the evaluation algorithm, whose complexity potentially depends on that of the constrain. The laconic property breaks this circular dependency.

Thus, we obtain a simple and direct obfuscation candidate that is not based on the functional encryption-to-obfuscation paradigm [AJ15, BV15a], and that is genuinely different from other lattice-based obfuscation candidates, e.g., [WW21, GP21, BDGM22].

We stress that, contrary to our laconic shift-hiding PRF, we do not know how to reduce the security of this construction to decomposed LWE (or any other standard assumption) and we leave a rigorous analysis of this scheme as a fascinating open problem. Nevertheless, we manage to show that for certain simple pseudorandom functionalities, the approach leads to provably secure obfuscation. This is the case, for instance, for circuits that compute lattice preimages using a hidden trapdoor.

### 1.2 Concurrent and Subsequent Work

In a concurrent work [GY26b], Goyal and Yadugiri show how to build adaptively secure (distributed) broadcast encryption in the standard model from decomposed LWE, using different techniques. Their adaptive construction achieves similar efficiency as ours, however, unlike our construction, their CRS size grows proportionally to $N$ even in the semi-static setting. In [GY26a] the same set of authors present a construction of flexible and identity-based broadcast encryption, with a security proof in the random oracle model.

Our tree encodings were used in a subsequent work [AMR26] which established the existence of correlation intractable hash functions for all (possibly inefficient) batched relations, assuming the hardness of decomposed SIS, the natural dual version of decomposed LWE.

### 1.3 Technical Outline: Tree Encodings

First, let us make our objective concrete. We want to construct algorithms $\mathsf{TreeEvalM}$, $\mathsf{TreeEvalH}$, and $\mathsf{TreeExpand}$ with the following syntax

- $\mathsf{TreeEvalM}(\mathsf{pk}, f) \mapsto \mathbf{M}_f$
- $\mathsf{TreeEvalH}(\mathsf{pk}, f, \alpha) \mapsto \mathbf{H}_{f,\alpha}$
- $\mathsf{TreeExpand}(\mathsf{pk}, \alpha) \mapsto \overline{\mathbf{P}}_\alpha$

such that for all $\delta$-depth boolean circuits $f$ (with $\delta \geq 1$) and all inputs $\alpha \in \{0,1\}^d$ it holds that

$$\mathbf{G}(\mathbf{M}_f \overline{\mathbf{P}}_\alpha - f(\alpha)^\intercal \otimes \mathbf{I}_n) = \mathbf{Z}\mathbf{H}_{f,\alpha}. \tag{3}$$

Moreover $\|\mathbf{H}_{f,\alpha}\|_\infty \leq \mathsf{poly}(\sigma,\lambda)^{d\delta}$ and $\|\overline{\mathbf{P}}_\alpha\|_\infty \leq \mathsf{poly}(\lambda)^d$. This equation is syntactically equivalent to the one introduced by Wee [Wee25], except for a crucial difference: The *circuit depth* of $\mathsf{TreeExpand}(\alpha)$ is bounded by $\log d \cdot O(\log\log q) = \mathsf{poly}(\log\lambda, \log d, \log\delta)$, where $q$ is the modulus. Note that the algorithm $\mathsf{TreeExpand}$ itself belongs to the class of bounded depth circuits supported by our encodings. As it will become clear in Section 1.4, this will be the main leverage for our applications. In what follows, we explain how to construct polynomial-time algorithms satisfying Equation (3).

**Read-Once Matrix Branching Programs.** A useful computational abstraction to understand our algorithms is *read-once matrix branching programs*, which we recall next. A read-once matrix branching program $\mathsf{BP}$ consists of two transition matrices $\mathbf{M}_{i,0}, \mathbf{M}_{i,1}$ for each input position $i \in [d]$, and evaluates to

$$\mathsf{BP}(\alpha) = \prod_{i\in[d]} \mathbf{M}_{i,\alpha_i}$$

on input $\alpha \in \{0,1\}^d$. The matrices may be rectangular, with matching dimensions at consecutive positions. We write $\dim(\mathsf{BP})$ for the largest number of rows or columns of a transition matrix. In this overview, we only consider branching programs with scalar output, so the first transition has one row and the last has one column.

An important fact about matrix branching programs is that this class is closed under scalar addition and multiplication. For addition, given scalar-output programs $\mathsf{BP}_0 = \{\mathbf{M}_{i,b}\}_{i,b}$ and $\mathsf{BP}_1 = \{\mathbf{N}_{i,b}\}_{i,b}$, define

$$\mathsf{BP}_2(\alpha) := \begin{bmatrix}\mathbf{M}_{0,\alpha_0} & \mathbf{N}_{0,\alpha_0}\end{bmatrix} \prod_{i=1}^{d-2} \begin{bmatrix}\mathbf{M}_{i,\alpha_i} & \mathbf{0} \\ \mathbf{0} & \mathbf{N}_{i,\alpha_i}\end{bmatrix} \begin{bmatrix}\mathbf{M}_{d-1,\alpha_{d-1}} \\ \mathbf{N}_{d-1,\alpha_{d-1}}\end{bmatrix}.$$

Multiplying the blocks gives

$$\mathsf{BP}_2(\alpha) = \mathsf{BP}_0(\alpha) + \mathsf{BP}_1(\alpha), \qquad \dim(\mathsf{BP}_2) \leq \dim(\mathsf{BP}_0) + \dim(\mathsf{BP}_1).$$

Multiplication is implemented by taking the tensor product of the two transitions at each position. Indeed,

$$\prod_{i\in[d]} (\mathbf{M}_{i,\alpha_i} \otimes \mathbf{N}_{i,\alpha_i}) = \left(\prod_{i\in[d]} \mathbf{M}_{i,\alpha_i}\right) \otimes \left(\prod_{i\in[d]} \mathbf{N}_{i,\alpha_i}\right) = \mathsf{BP}_0(\alpha)\mathsf{BP}_1(\alpha).$$

The first equality is its mixed-product identity, and the second uses that the outputs are scalars. The dimension of the product program is at most $\dim(\mathsf{BP}_0)\dim(\mathsf{BP}_1)$.

**Hashing Branching Programs.** Next, we recall the SIS-based linear hash function [AMR25a]. In the simple 2-to-1 settings, the function is a simple linear map

$$\mathsf{Hash}(\mathsf{pk}, \mathbf{x}) = \mathbf{D}\mathbf{x}, \qquad \mathbf{D} = \begin{bmatrix}\mathbf{D}_0 & \mathbf{D}_1\end{bmatrix}.$$

Here a digest is the vector returned by the hash, and $\mathbf{D}_0, \mathbf{D}_1 \in \mathbb{Z}_q^{mn \times mn}$, so the hash compresses two vectors of length $mn$ into one. To achieve better compression, the natural approach is to define a Merkle hash for a binary tree: If the two child digests of a node are $\mathbf{h}_0$ and $\mathbf{h}_1$, then the digest at the node is $\mathbf{D}_0\mathbf{h}_0 + \mathbf{D}_1\mathbf{h}_1$. Repeating this rule from the leaves to the root gives

$$\mathsf{MerkleHash}(\mathsf{pk}, \{\mathbf{x}_\alpha\}_\alpha) = \sum_{\alpha \in \{0,1\}^d} \left( \prod_{i \in [d]} \mathbf{D}_{\alpha_i} \right) \mathbf{x}_\alpha,$$

where $\mathbf{x}_\alpha$ is the vector at leaf $\alpha$.

Now imagine for a moment that we wanted to simultaneously compute the hash of *all possible outputs* of a matrix branching program $\{\mathsf{BP}(\alpha)\}_\alpha$. If done naively, this would take time at least $2^d$. Our first important observation is that the hash function defined above allows us to compute such a hash in time *polynomial* in $d$. We embed each scalar output as the first coordinate of a leaf vector, writing $\mathbf{e}_0$ for the first standard basis vector of $\mathbb{Z}_q^{mn}$. Expanding the expression for the root of the Merkle tree:

$$\begin{aligned} \mathsf{MerkleHash}\left(\mathsf{pk}, \{\mathsf{BP}(\alpha)\mathbf{e}_0\}_\alpha\right) &= \sum_\alpha \left( \prod_{i \in [d]} \mathbf{D}_{\alpha_i} \right) \mathbf{e}_0 \mathsf{BP}(\alpha) \\ &= \left( \prod_{i \in [d]} \left( \mathbf{D}_0 \otimes \mathbf{M}_{i,0} + \mathbf{D}_1 \otimes \mathbf{M}_{i,1} \right) \right) \mathbf{e}_0. \end{aligned}$$

The second equality follows by substituting the definition of the branching program and applying the mixed-product identity of tensor products. Clearly, there is an algorithm computing the expression in the last line running in time polynomial in $d$, in $\dim(\mathsf{BP})$, and in the public-parameter sizes. The same computation gives any subtree digest, by fixing the input prefix that selects that subtree.

The hash function from [AMR25a] also supports the following expansion equation. Let $\mathbf{P} = [\mathbf{P}_0 \mid \mathbf{P}_1]$ be a low-norm matrix fixed by the public parameters. Let $\mathbf{h}_\rho$ be the digest at node $\rho$, with children $\rho 0, \rho 1$. If $\mathbf{y}_\rho$ concatenates their two digests, then

$$\mathbf{G}(\mathbf{h}_\rho^\intercal \otimes \mathbf{I}_n)\mathbf{P}_b - \mathbf{Z}\mathbf{Q}_{\mathbf{y}_\rho, b} = \mathbf{G}(\mathbf{h}_{\rho b}^\intercal \otimes \mathbf{I}_n).$$

Here $b \in \{0,1\}$ selects a child, $\mathbf{G}$ is the public gadget matrix that recombines binary digits, $\mathbf{I}_n$ is the identity matrix, and $\mathbf{Z}$ is a public matrix supplied by setup. The low-norm matrix $\mathbf{Q}_{\mathbf{y}_\rho, b}$ is efficiently computed from the two child digests by the encoding-expansion algorithm. Applying this equation along the path to $\alpha$ gives

$$\mathbf{G}\left(\mathsf{MerkleHash}(\mathsf{pk}, \{\mathsf{BP}(\beta)\mathbf{e}_0\}_\beta)^\intercal \otimes \mathbf{I}_n\right) \left( \prod_{i \in [d]} \mathbf{P}_{\alpha_i} \right) (\mathbf{e}_0 \otimes \mathbf{I}_n) + \mathbf{Z}\mathbf{T}_{\mathsf{BP},\alpha} = \mathsf{BP}(\alpha)\mathbf{G}. \tag{4}$$

The matrix $\mathbf{T}_{\mathsf{BP},\alpha}$ collects the low-norm corrections along this path; the final factor $\mathbf{e}_0 \otimes \mathbf{I}_n$ selects the coordinate holding the scalar output. Each correction only requires subtree digests, so the hashing identity above makes the entire expansion efficient without constructing the exponentially large tree.

**Warm Up: Projection Functions.** We have now all the elements needed to understand the algorithms satisfying Equation (3) for the special case of projection functions $\Pi_i(\alpha) = \alpha_i$. First, notice that such a function can be computed by a matrix branching program $\mathsf{BP}_i$ whose transitions are all [1], except that $\mathbf{M}_{i,0} = [0]$. As explained above, we can efficiently compute

$$\mathbf{h}_i := \mathsf{MerkleHash}(\mathsf{pk}, \{\mathsf{BP}_i(\beta)\mathbf{e}_0\}_\beta), \qquad \mathbf{M}_i := \mathbf{G}^{-1}(\mathbf{G}(\mathbf{h}_i^\intercal \otimes \mathbf{I}_n)) \,.$$

Here $\mathbf{G}^{-1}$ denotes columnwise binary decomposition, so $\mathbf{G}\mathbf{M}_i = \mathbf{G}(\mathbf{h}_i^\intercal \otimes \mathbf{I}_n)$. Define

$$\mathsf{MBP}_{\Pi_i}(\alpha) := \mathbf{M}_i \underbrace{\left(\prod_{j\in[d]} \mathbf{P}_{\alpha_j}\right)(\mathbf{e}_0 \otimes \mathbf{I}_n)}_{=:\hat{\mathbf{P}}_\alpha}, \qquad \mathbf{H}_{\Pi_i,\alpha} := -\mathbf{T}_{\mathsf{BP}_i,\alpha}.$$

By Equation (4), we have

$$\mathbf{G}(\mathbf{M}_i\hat{\mathbf{P}}_\alpha) - \mathbf{Z}\mathbf{H}_{\Pi_i,\alpha} = \mathbf{G}\mathsf{MBP}_{\Pi_i}(\alpha) - \mathbf{Z}\mathbf{H}_{\Pi_i,\alpha} = \alpha_i\mathbf{G}$$

as desired. The same construction with every scalar transition equal to [1] gives the constant-one function. An observation that will be useful in a moment is that $\mathsf{MBP}_{\Pi_i}$ is itself a matrix branching program: Absorb $\mathbf{M}_i$ and $(\mathbf{e}_0 \otimes \mathbf{I}_n)$ into its first and last transitions to obtain the matrices that define it.

**Homomorphic Computation.** We would like now to extend our algorithms from projections to all arithmetic circuits. For every intermediate function $f : \{0,1\}^d \to \mathbb{Z}_q$, we will maintain a matrix branching program $\mathsf{MBP}_f$ satisfying the following invariant:

$$\mathbf{G}\mathsf{MBP}_f(\alpha) - \mathbf{Z}\mathbf{H}_{f,\alpha} = f(\alpha)\mathbf{G}. \tag{5}$$

Here $\mathsf{MBP}_f$ has $n \times n$ matrix outputs, and $\mathbf{H}_{f,\alpha}$ is an efficiently computable low-norm correction. We have just seen that Equation (5) is satisfied for input gates, so we proceed inductively over addition and multiplication gates.

For addition, set

$$\mathsf{MBP}_{f+g}(\alpha) := \mathsf{MBP}_f(\alpha) + \mathsf{MBP}_g(\alpha), \qquad \mathbf{H}_{f+g,\alpha} := \mathbf{H}_{f,\alpha} + \mathbf{H}_{g,\alpha}.$$

Note that the description of the matrix branching program $\mathsf{MBP}_{f+g}$ can be obtained by composing $\mathsf{MBP}_f$ and $\mathsf{MBP}_g$, as explained above. Correctness follows by adding the two invariants:

$$\begin{aligned}\mathbf{G}\mathsf{MBP}_{f+g}(\alpha) - \mathbf{Z}\mathbf{H}_{f+g,\alpha} &= (\mathbf{G}\mathsf{MBP}_f(\alpha) - \mathbf{Z}\mathbf{H}_{f,\alpha}) + (\mathbf{G}\mathsf{MBP}_g(\alpha) - \mathbf{Z}\mathbf{H}_{g,\alpha}) \\ &= (f(\alpha) + g(\alpha))\mathbf{G}.\end{aligned}$$

For multiplication, set

$$\mathsf{MBP}_{fg}(\alpha) := \mathsf{MBP}_f(\alpha)\mathsf{MBP}_g(\alpha), \qquad \mathbf{H}_{fg,\alpha} := \mathbf{H}_{f,\alpha}\mathsf{MBP}_g(\alpha) + f(\alpha)\mathbf{H}_{g,\alpha}.$$

The product program is efficiently constructed using the same closure properties: Each entry of a matrix product is a sum of products of scalar entries. Correctness follows from

$$\begin{aligned}\mathbf{G}\mathsf{MBP}_{fg}(\alpha) - \mathbf{Z}\mathbf{H}_{fg,\alpha} &= (\mathbf{G}\mathsf{MBP}_f(\alpha) - \mathbf{Z}\mathbf{H}_{f,\alpha})\mathsf{MBP}_g(\alpha) - f(\alpha)\mathbf{Z}\mathbf{H}_{g,\alpha} \\ &= f(\alpha)(\mathbf{G}\mathsf{MBP}_g(\alpha) - \mathbf{Z}\mathbf{H}_{g,\alpha}) \\ &= f(\alpha)g(\alpha)\mathbf{G}.\end{aligned}$$

For a constant $c \in \mathbb{Z}_q$, right-multiplying both $\mathsf{MBP}_f(\alpha)$ and $\mathbf{H}_{f,\alpha}$ by $\mathbf{G}^{-1}(c\mathbf{G})$ similarly gives the invariant for $cf$. Together with the constant-one program, this also supplies constant wires.

There is, however, a problem with iterating multiplication: The dimensions of the composed programs multiply. More precisely, we have

$$\dim(\mathsf{MBP}_{fg}) \leq n \dim(\mathsf{MBP}_f) \dim(\mathsf{MBP}_g).$$

Thus repeated multiplication of programs of comparable dimension can square the dimension at each circuit layer. To keep the algorithms efficient, we need a way to *flatten* the program after each gate, resetting its dimension while preserving Equation (5).

**Flattening.** The idea is to apply hashing and expansion once more, except now to compute the entries of the output of $\mathsf{MBP}_f$. For $r, s \in [n]$, let $\mathsf{MBP}_{r,s}(\alpha)$ denote its $(r,s)$-entry, which is also computed by a matrix branching program $\mathsf{MBP}_{r,s}$ that selects row $r$ of the first transition and column $s$ of the last. Again, we can efficiently compute

$$\mathbf{h}_{r,s} := \mathsf{MerkleHash}(\mathsf{pk}, \{\mathsf{MBP}_{r,s}(\beta)\mathbf{e}_0\}_\beta).$$

Applying Equation (4) to each of these scalar programs gives

$$\mathbf{G}(\mathbf{h}_{r,s}^\intercal \otimes \mathbf{I}_n) \left( \prod_{i \in [d]} \mathbf{P}_{\alpha_i} \right) (\mathbf{e}_0 \otimes \mathbf{I}_n) + \mathbf{Z}\mathbf{T}_{r,s,\alpha} = \mathsf{MBP}_{r,s}(\alpha)\mathbf{G}, \tag{6}$$

where $\mathbf{T}_{r,s,\alpha}$ is the correction for that entry. Crucially, the matrices inside the product are the same for all entries and all programs; only the hashes and corrections depend on $\mathsf{MBP}_f$.

What is left is to collect these equations into a single matrix equation. Let $\mathbf{e}_r, \mathbf{e}_s$ be the corresponding standard basis vectors of $\mathbb{Z}_q^n$, and let $\mathbf{R}$ be the vertical stack of the blocks $(\mathbf{e}_0 \otimes \mathbf{I}_n)\mathbf{e}_r\mathbf{e}_s^\intercal$. In the same order, concatenate the hash matrices horizontally and set

$$\mathbf{M}_f := \mathbf{G}^{-1}\Big(\mathbf{G}\left[\mathbf{h}_{r,s}^\intercal \otimes \mathbf{I}_n\right]_{r,s \in [n]}\Big), \qquad \overline{\mathbf{P}}_\alpha := \left( \prod_{i \in [d]} (\mathbf{I}_{n^2} \otimes \mathbf{P}_{\alpha_i}) \right) \mathbf{R}.$$

The factor $\mathbf{I}_{n^2} \otimes \mathbf{P}_{\alpha_i}$ runs one copy of the expansion for each matrix entry, and $\mathbf{R}$ places the expanded entries in their original positions. Define the flattened program and its correction by

$$\widehat{\mathsf{MBP}}_f(\alpha) := \mathbf{M}_f\overline{\mathbf{P}}_\alpha, \qquad \mathbf{H}^{\mathsf{flat}}_{\mathsf{MBP}_f,\alpha} := -\sum_{r,s \in [n]} \mathbf{T}_{r,s,\alpha}\mathbf{e}_r\mathbf{e}_s^\intercal.$$

Multiplying Equation (6) on the right by $\mathbf{e}_r\mathbf{e}_s^\intercal$ and summing over the entries proves

$$\mathbf{G}\widehat{\mathsf{MBP}}_f(\alpha) - \mathbf{Z}\mathbf{H}^{\mathsf{flat}}_{\mathsf{MBP}_f,\alpha} = \mathbf{G}\mathsf{MBP}_f(\alpha). \tag{7}$$

Consequently, adding $\mathbf{H}^{\mathsf{flat}}_{\mathsf{MBP}_f,\alpha}$ to $\mathbf{H}_{f,\alpha}$ preserves the invariant. The new matrix program $\widehat{\mathsf{MBP}}_f$ has dimension at most $mn^4$, independent of its dimension before flattening: Each $\mathbf{P}_b$ has dimension $mn^2$, and we use $n^2$ copies. Applying this procedure after every gate therefore keeps the computation efficient.

To summarize, $\mathsf{TreeEvalM}$ returns $\mathbf{M}_f$, $\mathsf{TreeExpand}$ returns $\overline{\mathbf{P}}_\alpha$, and $\mathsf{TreeEvalH}$ computes the correction accumulated through the gates and flattening steps. Renaming this final correction $\mathbf{H}_{f,\alpha}$, the invariant becomes

$$\mathbf{G}(\mathbf{M}_f\overline{\mathbf{P}}_\alpha - f(\alpha)\mathbf{I}_n) = \mathbf{Z}\mathbf{H}_{f,\alpha}$$

which is exactly Equation (3). For a fixed output length, $\overline{\mathbf{P}}_\alpha$ depends only on the public parameters and $\alpha$. Its $d$ selected matrices can be multiplied in pairs, then in pairs of products, forming a balanced tree with $O(\log d)$ layers. This gives the low circuit depth of $\mathsf{TreeExpand}$ stated above.

### 1.4 Technical Outline: Applications

We give an overview of the new applications enabled by our tree encodings.

**Laconic Shift-Hiding PRF.** In what follows, we assume familiarity with the notion of laconic function evaluation (LFE) [QWW18]. Our shift-hiding PRF [PS18] consists of the following algorithms:

- (Setup) We define
$$g_{\mathsf{crs}}(\alpha) := \mathsf{vec}(\mathsf{LFE.Hash}(\mathsf{crs}, f_\alpha)),$$
where $f_\alpha$ is the shift function with $\alpha$ hardwired. We run $\mathsf{TreeEvalM}(\mathsf{pk}, g_{\mathsf{crs}})$ to obtain $\tilde{\mathbf{M}}$. The key consists of $\tilde{\mathbf{M}}$, a randomly sampled $\mathbf{s}$, a random mask $\mathbf{t}$, $\mathbf{z}^\intercal \approx \mathbf{s}^\intercal\mathbf{Z}$, and the public parameters.

- (Shift) On a shift description $\beta \in \{0,1\}^\eta$, compute $\mathsf{ec} \xleftarrow{\$} \mathsf{LFE.Encode}(\mathsf{crs}, \mathbf{s}, \beta)$ and output the shifted key containing $(\mathsf{pk}, \mathsf{crs}, \mathbf{z}, \mathsf{ec}, \mathbf{t})$.

- (Eval) On input $\alpha$, the regular evaluation algorithm computes
$$\left\lceil \mathbf{s}^\intercal\mathbf{G}\tilde{\mathbf{M}} \cdot \overline{\mathbf{P}}_\alpha \cdot \mathbf{V}' + \mathbf{t}^\intercal \right\rfloor_p$$
where $\overline{\mathbf{P}}_\alpha = \mathsf{TreeExpand}(\mathsf{pk}, \alpha)$ and $\mathbf{V}'$ is the fixed selector that maps the vectorized LFE hash back to a matrix. We use $\lceil\cdot\rfloor_p$ to denote rounding onto $\mathbb{Z}_p$ for a smaller modulus $p$.

- (Shifted Eval) On input $\alpha$, we derive $\mathbf{H}_{g_{\mathsf{crs}},\alpha} = \mathsf{TreeEvalH}(\mathsf{pk}, g_{\mathsf{crs}}, \alpha)$. The output is set to be
$$\left\lceil \mathsf{LFE.Eval}(\mathsf{crs}, \mathsf{ec}, f_\alpha) + \mathbf{z}^\intercal \cdot \mathbf{H}_{g_{\mathsf{crs}},\alpha} \cdot \mathbf{V}' + \mathbf{t}^\intercal \right\rfloor_p.$$

It is easy to show that the shifted key computationally hides the function $f$, with a direct reduction to the decomposed LWE assumption. The more interesting bit is to show correctness of the shifted eval algorithm. Indeed, observe that

$$\begin{aligned}
&\mathsf{LFE.Eval}(\mathsf{crs}, \mathsf{ec}, f_\alpha) + \mathbf{z}^\intercal \cdot \mathbf{H}_{g_{\mathsf{crs}},\alpha} \cdot \mathbf{V}' \\
&\approx \mathbf{s}^\intercal\mathbf{G}\mathsf{LFE.Hash}(\mathsf{crs}, f_\alpha) + \frac{q}{p}f(\alpha,\beta) + \mathbf{z}^\intercal \cdot \mathbf{H}_{g_{\mathsf{crs}},\alpha} \cdot \mathbf{V}' \\
&\approx \mathbf{s}^\intercal\mathbf{G}\mathsf{LFE.Hash}(\mathsf{crs}, f_\alpha) + \frac{q}{p}f(\alpha,\beta) + \mathbf{s}^\intercal\mathbf{G}(\tilde{\mathbf{M}} \cdot \overline{\mathbf{P}}_\alpha - g_{\mathsf{crs}}(\alpha)^\intercal \otimes \mathbf{I}_n) \cdot \mathbf{V}' \\
&= \mathbf{s}^\intercal\mathbf{G}\tilde{\mathbf{M}} \cdot \overline{\mathbf{P}}_\alpha \cdot \mathbf{V}' + \frac{q}{p}f(\alpha,\beta)
\end{aligned}$$

where the first identity follows by the correctness of the LFE, the second by the key equation, and the third by the definition of $\mathbf{V}'$. The same mask $\mathbf{t}$ is added by both evaluation algorithms. Ignoring the noise terms, which are removed by rounding, the shifted evaluation therefore returns the regular evaluation shifted by $f(\alpha, \beta)$.

The crucial property of such a construction is that it is *laconic*: The (non-shifted) evaluation algorithm only depends on a bound on the depth of $f$. In particular, the depth of the (non-shifted) evaluation grows polylogarithmically in the depth of $f$ and its input size. As we shall see shortly, this is the property that will enable important applications as (ignoring the security aspect) nothing prevents us from shifting a key by its own evaluation.

**A Heuristic Approach to Obfuscation.** In a laconic shift-hiding function, the evaluation of an unshifted key has lower complexity than the evaluation of a shifted key. Thus, it is possible, in principle, to evaluate the unshifted PRF inside the shift. This leads to the following heuristic approach to obfuscation: To obfuscate a circuit $C$, we first run the setup algorithm to compute a PRF key $K$, then we define the function $g_K$ to be

$$g_K(x) = C(x) - \mathsf{Eval}(K, x)$$

where $\mathsf{Eval}$ is the unshifted evaluation algorithm. The obfuscation will be $\tilde{K} \xleftarrow{\$} \mathsf{Shift}(K, g_K)$. One can see that evaluating the obfuscated circuit returns the correct result, since

$$\begin{aligned}\mathsf{ShiftEval}(\widetilde{K}, x) &= \mathsf{Eval}(K, x) + g_K(x) \\ &= \mathsf{Eval}(K, x) + C(x) - \mathsf{Eval}(K, x) \\ &= C(x).\end{aligned}$$

Of course, the circularity we introduce does not allow us to prove this construction secure, but we could not disprove it either. If this approach manages to withstand future cryptanalysis, this approach could provide a more direct way to construct obfuscation, avoiding the large computational overheads introduced by recursion in the current iO constructions [BV15a, AJ15]. In the meantime, we advise caution.

**Pseudorandom Obfuscation.** Although the circularity prevents us from proving the security of our general-purpose iO construction, we shall see how we can circumvent this barrier, for specific applications. The first is a compiler that turns a regular iO obfuscator $\mathsf{Obf}$ into a pseudorandom one. We recall that a pseudorandom obfuscator guarantees that the obfuscation of a random PRF is indistinguishable from a random string [BDJ+25]. We show that our construction satisfies this property if the PRF is puncturable and subexponentially secure. To obfuscate a circuit $C$, we first derive $\tilde{g} = \mathsf{Obf}(1^\lambda, g_K)$, and then output $\mathsf{Shift}(K, \tilde{g})$. The correctness of the construction follows from the correctness of $\mathsf{Obf}$ and of the shift-hiding function, as shown above. For security, notice that if $C(x)$ computes the evaluation of a random puncturable PRF, one can apply a puncturing argument to gradually remove all information about $K$ from $\tilde{g}$ [SW14], at which point pseudorandomness follows directly from the shift-hiding property.

**Obfuscating Lattice Preimage Sampling.** We show that the heuristic approach of evaluating the unshifted key inside the shift leads to a *provably secure obfuscation* for the lattice preimage

sampling procedure [MP12], in a sense that we will clarify shortly. Specifically, we can construct an obfuscation for the functionality $C$ that on input a matrix $\mathbf{M}$ and auxiliary information $\mathsf{aux}$ proceeds as follows:

- Uses some hardwired secret $\tau$ to extract a lattice trapdoor $\mathbf{T}$ for $\mathbf{M}$ from $\mathsf{aux}$.
- Uses $\mathbf{T}$ to generate a random low-norm vector $\tilde{\mathbf{w}}$ such that $\mathbf{M} \cdot \tilde{\mathbf{w}} = \mathbf{Q} \cdot \mathbf{y}_{\mathbf{M},\mathsf{aux}}$, where $\mathbf{y}_{\mathbf{M},\mathsf{aux}}$ is a low-norm vector and $\mathbf{Q}$ is a random matrix hardcoded in the circuit.
- Outputs $\tilde{\mathbf{w}}$.

As we will see later, this functionality is very useful: It ensures that if we provide an LWE sample $\mathbf{s}^\intercal \cdot \mathbf{M} \cdot \tilde{\mathbf{w}} + e$, where $\tilde{\mathbf{w}}$ is the output of the obfuscated program, the value of $\mathbf{s}$ remains secret. Indeed, we can apply LWE with respect to the random matrix $\mathbf{Q}$ as

$$\mathbf{s}^\intercal \mathbf{M} \cdot \tilde{\mathbf{w}} \approx \mathbf{s}^\intercal \mathbf{Q} \cdot \mathbf{y}_{\mathbf{M},\mathsf{aux}}.$$

To understand how we achieve this, we need to recall how the preimage sampling procedure of [MP12] works. A trapdoor $\mathbf{T}$ for a matrix $\mathbf{M}$ corresponds to a low-norm matrix such that $\mathbf{M} \cdot \mathbf{T} = \mathbf{G}$. The goal of preimage sampling is to derive, given as input a vector $\mathbf{u}$, a *low-norm* vector $\mathbf{w}$ such that $\mathbf{M} \cdot \mathbf{w} = \mathbf{u}$. The (simplified) idea of [MP12] is the following: First sample a random low norm vector $\mathbf{w}'$, then output $\mathbf{w} := \mathbf{w}' + \mathbf{T} \cdot \mathbf{G}^{-1}(\mathbf{u} - \mathbf{M} \cdot \mathbf{w}')$. It is easy to see that

$$\mathbf{M} \cdot \mathbf{w} = \mathbf{M} \cdot \mathbf{w}' + \mathbf{M} \cdot \mathbf{T} \cdot \mathbf{G}^{-1}(\mathbf{u} - \mathbf{M} \cdot \mathbf{w}') = \mathbf{u}.$$

Moreover, if $\mathbf{w}'$ is sampled from a suitable low-norm distribution, the output $\mathbf{w}$ looks like a random low-norm sample, conditioned on $\mathbf{M} \cdot \mathbf{w} = \mathbf{u}$. Contrary to [MP12], where $\mathbf{w}'$ is sampled from a discrete Gaussian, here we sample it from the uniform distribution over some bounded interval $[-p/2, p/2]$.

To obfuscate $C$ we use two shift-hiding functions: The first $\mathsf{SH}_q$ outputs vectors over $\mathbb{Z}_q$ where $q \gg p$ is the modulus over which $\mathbf{M}$ is defined, the second one, $\mathsf{SH}_p$, outputs vectors over $\mathbb{Z}_p$. Let $K_0$ be the unshifted key for $\mathsf{SH}_q$ and let $K_1$ be the unshifted key for $\mathsf{SH}_p$. The obfuscation algorithm defines the shifted keys as:

- $\tilde{K}_0 \xleftarrow{\$} \mathsf{SH}_q.\mathsf{Shift}(K_0, g_0)$, where $g_0(\mathbf{M}, \mathsf{aux}) = -\mathbf{M} \cdot \mathbf{w} + g'(\mathbf{M}, \mathsf{aux})$, $g'$ is a function we are going to define soon, and $\mathbf{w} := \mathsf{SH}_p.\mathsf{Eval}(K_1, (\mathbf{M}, \mathsf{aux}))$.
- $\tilde{K}_1 \xleftarrow{\$} \mathsf{SH}_p.\mathsf{Shift}(K_1, g_1)$, where $g_1(\mathbf{M}, \mathsf{aux}) = \mathbf{T} \cdot \mathbf{G}^{-1}(\tilde{\mathbf{u}})$ and $\tilde{\mathbf{u}} = \mathsf{SH}_q.\mathsf{ShiftEval}(\tilde{K}_0, (\mathbf{M}, \mathsf{aux}))$.

The obfuscation consists of the shifted key $\tilde{K}_1$.[1]

To evaluate the obfuscation on input $(\mathbf{M}, \mathsf{aux})$, we just evaluate the shifted key to obtain $\tilde{\mathbf{w}} \in \mathbb{Z}_p$. To see why this constitutes a valid output of $C$, notice that [2]

$$\begin{aligned}
\mathbf{M} \cdot \tilde{\mathbf{w}} &= \mathbf{M} \cdot (\mathbf{w} + \mathbf{T} \cdot \mathbf{G}^{-1}(\tilde{\mathbf{u}})) \\
&= \mathbf{M} \cdot \mathbf{w} + \tilde{\mathbf{u}} \\
&= \mathsf{SH}_q.\mathsf{Eval}(K_0, (\mathbf{M}, \mathsf{aux})) + g'(\mathbf{M}, \mathsf{aux}).
\end{aligned}$$

[1] Observe that the circularity in the definition of the shifts is only well-defined because of the laconic property of the shift-hiding PRFs.

[2] In the above computations, we are actually mixing modulo $q$ and modulo $p$ computations. There is the risk that wrap arounds compromise correctness, however, if we pick $p$ to be significantly larger than the norm of the trapdoor $\mathbf{T}$, the probability of this event can be made negligible.

Now, let $f_{\mathbf{M},\mathsf{aux}}$ be the function $K_0 \mapsto \mathsf{SH}_q.\mathsf{Eval}(K_0, (\mathbf{M}, \mathsf{aux}))$ and define

$$\begin{aligned}
\mathbf{u} &:= \mathsf{SH}_q.\mathsf{Eval}(K_0, (\mathbf{M}, \mathsf{aux})) \\
\mathbf{Q}' &:= \mathbf{Q} - \mathsf{Bits}(K_0)^\intercal \otimes \mathbf{G} \\
\mathbf{K}_{\mathbf{M},\mathsf{aux}} &:= \mathsf{BGG}^+.\mathsf{EvalM}(f_{\mathbf{M},\mathsf{aux}}, \mathbf{Q}') \\
\mathbf{H}_{\mathbf{M},\mathsf{aux}} &:= \mathsf{BGG}^+.\mathsf{EvalC}(f_{\mathbf{M},\mathsf{aux}}, \mathbf{Q}', K_0).
\end{aligned}$$

Check what happens if we define

$$g'(\mathbf{M}, \mathsf{aux}) := \mathbf{Q}' \cdot \mathbf{K}_{\mathbf{M},\mathsf{aux}} \cdot \mathbf{v}$$

where $\mathbf{v}$ is the low-norm vector such that, for every $\mathbf{y} \in \mathbb{Z}_q^k$, we have $(\mathbf{G}^{-1}(\mathbf{y})^\intercal \otimes \mathbf{G}) \cdot \mathbf{v} = \mathbf{y}$. By the key equation of [BGG+14], we have that

$$\underbrace{(\mathbf{Q}' + \mathsf{Bits}(K_0)^\intercal \otimes \mathbf{G})}_{\mathbf{Q}} \cdot \mathbf{H}_{\mathbf{M},\mathsf{aux}} = \mathbf{Q}'\mathbf{K}_{\mathbf{M},\mathsf{aux}} + \mathbf{G}^{-1}(\mathbf{u})^\intercal \otimes \mathbf{G}$$

Therefore, if we multiply both sides by $\mathbf{v}$, we obtain that

$$\mathbf{Q} \cdot \mathbf{H}_{\mathbf{M},\mathsf{aux}} \cdot \mathbf{v} = \mathbf{u} + g'(\mathbf{M}, \mathsf{aux}).$$

We have just proven that $\mathbf{M} \cdot \tilde{\mathbf{w}} = \mathbf{Q} \cdot \mathbf{h}_{\mathbf{M},\mathsf{aux}}$, where $\mathbf{h}_{\mathbf{M},\mathsf{aux}}$ is a low-norm vector.

For security, we show that $\tilde{K}_1$ is indistinguishable from uniform. We start by observing that $\mathbf{Q}'$ does not leak any information about $K_0$ as $\mathbf{Q}$ masks all the information. Next, we apply the shift hiding property to substitute $\tilde{K}_0$ with a random string (we can do this because $g_0, g_1$ and $g'$ do not use $K_0$). Finally, it is the turn of $\tilde{K}_1$. The operation can be performed because now $\tilde{K}_0$ no longer depends on $K_1$.

**Compressing LWE Samples.** Next, we show a method to compress LWE samples, which is going to be instrumental for our broadcast encryption scheme. Let us make our task more concrete: We are given as input a matrix $\mathbf{X}$ and we want to compute a hash $\mathsf{Hash}(\mathbf{X})$ such that an LWE sample with respect to the hash can be publicly expanded

$$\mathsf{Expand}(\mathbf{s}^\intercal \mathsf{Hash}(\mathbf{X}) + \mathbf{e}^\intercal) \mapsto \mathbf{s}^\intercal \mathbf{X} + \tilde{\mathbf{e}}^\intercal$$

into an LWE sample with respect to $\mathbf{X}$. We show that if $\mathbf{X}$ satisfies additional properties, e.g., $\mathbf{X}$ is sampled at random, the sample $\mathbf{s}^\intercal \mathsf{Hash}(\mathbf{X}) + \mathbf{e}^\intercal$ also looks indistinguishable from random. We defer the description of the security notion to a later point in this overview.

The basic intuition is that by multiplying the decomposed LWE matrices by low-norm matrices and performing some restructuring, we are able to derive matrices $\mathbf{U} \in \mathbb{Z}_q^{k(t/n) \times t}$, $\mathbf{B} \in \mathbb{Z}_q^{(t/n) \times m}$, and $\mathbf{L} \in \mathbb{Z}_q^{mn \times t}$, where $\mathbf{B}$ and $\mathbf{L}$ are low-norm, $t > mn$, and

$$\left[\begin{array}{c|c} \mathbf{U} & \mathbf{B} \otimes \mathbf{G} \end{array}\right] \cdot \begin{bmatrix} \mathbf{I}_t \\ \mathbf{L} \end{bmatrix} = \mathbf{I}_{t/n} \otimes \mathbf{G}. \tag{8}$$

In other words, $\mathbf{L}$ is a lattice trapdoor for $\left[\begin{array}{c|c} \mathbf{U} & \mathbf{B} \otimes \mathbf{G} \end{array}\right]$. Under the hardness of decomposed LWE, we can show that any sample $(\mathbf{I}_{t/n} \otimes \mathbf{s}^\intercal) \cdot \mathbf{U} + \mathbf{E}$, where $\mathbf{E}$ is a noise term, looks random. This fact was already implicit in the work of [CHW25].

Now, take $\mathbf{X} \in \mathbb{Z}_q^{k \times t/n}$ and use $\mathbf{L}$ to compute low-norm vectors $\mathbf{w}_0, \mathbf{w}_1$ such that

$$\left[ \begin{array}{c|c} \mathbf{U} & \mathbf{B} \otimes \mathbf{G} \end{array} \right] \cdot \begin{bmatrix} \mathbf{w}_0 \\ \mathbf{w}_1 \end{bmatrix} = \mathsf{vec}(\mathbf{X}).$$

The hash of $\mathbf{X}$ will consist of $(\mathbf{I}_m \otimes \mathbf{G}) \cdot \mathbf{w}_1$. Notice that this has size $mn$, while $\mathbf{X}$ has size $t$.

The compressed LWE sample consists of two components $\mathbf{h} \approx (\mathbf{I}_m \otimes \mathbf{s}^\intercal) \cdot (\mathbf{I}_m \otimes \mathbf{G}) \cdot \mathbf{w}_1$ and $\mathbf{C} \approx (\mathbf{I}_{t/n} \otimes \mathbf{s}^\intercal) \cdot \mathbf{U}$. To expand it, one can simply compute:

$$\begin{aligned} \mathbf{C} \cdot \mathbf{w}_0 + \mathbf{B} \cdot \mathbf{h} &\approx (\mathbf{I}_{t/n} \otimes \mathbf{s}^\intercal) \cdot \mathbf{U} \cdot \mathbf{w}_0 + \mathbf{B} \cdot (\mathbf{I}_m \otimes \mathbf{s}^\intercal) \cdot (\mathbf{I}_m \otimes \mathbf{G}) \cdot \mathbf{w}_1 \\ &= (\mathbf{I}_{t/n} \otimes \mathbf{s}^\intercal) \cdot \mathbf{U} \cdot \mathbf{w}_0 + (\mathbf{I}_{t/n} \otimes \mathbf{s}^\intercal) \cdot (\mathbf{B} \otimes \mathbf{I}_k) \cdot (\mathbf{I}_m \otimes \mathbf{G}) \cdot \mathbf{w}_1 \\ &= (\mathbf{I}_{t/n} \otimes \mathbf{s}^\intercal) \cdot (\mathbf{U} \cdot \mathbf{w}_0 + (\mathbf{B} \otimes \mathbf{G}) \cdot \mathbf{w}_1) \\ &= (\mathbf{I}_{t/n} \otimes \mathbf{s}^\intercal) \cdot \mathsf{vec}(\mathbf{X}) \\ &= \mathbf{X}^\intercal \cdot \mathbf{s}. \end{aligned}$$

There are however two caveats. Here is the first one: As we described it, the above algorithm does not compress at all! In fact, $\mathbf{C}$ has size $t^2/n$, which is much larger than the size of $\mathbf{X}$. We deal with this aspect using recursion, similarly to [AMR25b, AMR25a, Wee25]. We can start from a matrix $\mathbf{X}$ with a much larger number of columns. Then, we hash it using a Merkle tree where the basic hash function is the one we described above. Given $\mathbf{C}$ (whose dimension remained unvaried) and an LWE sample with respect to the hash at the root, we can efficiently derive an LWE sample with respect to $\mathbf{X}$ by performing the expansion procedure layer after layer.

The second caveat is the fact that, in the current description, the hashing procedure is randomized. This allows us to argue security: if $\mathbf{X}$ is randomly sampled, we can prove that $(\mathbf{I}_m \otimes \mathbf{G}) \cdot \mathbf{w}_1$ is a random vector. At this point, we could just appeal to the standard decomposed LWE assumption to argue that $\mathbf{s}$ remains hidden. The issue with a randomized construction is that to correctly re-expand the LWE sample, one needs to receive the randomness used by the hasher. This is as big as $\mathbf{X}$, defeating the whole purpose of the construction. To obviate this problem, we modify the above algorithm to sample $\mathbf{w}_0$ and $\mathbf{w}_1$ using a variant of our newly constructed *obfuscated preimage sampler*. Specifically, as part of the public parameters we will provide a shifted PRF key, which will be evaluated by the hasher (and the expander) on $\mathbf{X}$ to derive the randomness for the preimage sampling. We will show that we can simulate $\mathbf{X}$ and the shifted key so that $(\mathbf{I}_m \otimes \mathbf{G}) \cdot \mathbf{w}_1$ becomes a matrix for which we can securely provide LWE samples.

**Adaptive Broadcast Encryption.** Recall that the goal of broadcast encryption is to allow a sender to encrypt a message under the public keys of a subset of participants, so that only the members of this subset will be able to decrypt. The challenge is to make the ciphertext of size, ideally, independent of the subset. We show how to use the techniques introduced above to build broadcast encryption that remains secure even when the subset of recipients is chosen after seeing the public parameters and the public keys (this is known as the *semi-static* security settings).

In our construction, the public keys consist of matrices $\mathbf{Q}_i = \mathbf{D} \cdot \mathbf{R}_i$, where $\mathbf{D}$ is a random matrix included in the public parameters of the construction and $\mathbf{R}_i$ is a random low-norm matrix, known only to the $i$-th party. To compute an encryption of a message $\mu \in \{0, 1\}$, we first derive low-norm vectors $\mathbf{u}_0, \ldots, \mathbf{u}_{N-1}$ from a shifted PRF key, where $N$ is the number of recipients in the set. Next, we derive the matrix

$$\mathbf{X} = \left[ \begin{array}{c|c|c} \mathbf{x}_0 & \cdots & \mathbf{x}_{N-1} \end{array} \right]$$

where $\mathbf{x}_i := \left[\begin{array}{c|c} \mathbf{Q}_i & \mathbf{D} \end{array}\right] \cdot \mathbf{u}_i$ and $\mathbf{Q}_i$ are the public keys belonging to the $i$-th party in the set of recipients. The ciphertext consists of two components:

- A compressed LWE sample with respect to $\mathbf{X}$, which consists of $\mathbf{C} \approx (\mathbf{I}_{t/n} \otimes \mathbf{s}^\intercal) \cdot \mathbf{U}$ and $\mathbf{h} \approx (\mathbf{I}_m \otimes \mathbf{s}^\intercal) \cdot \mathsf{Hash}(\mathbf{X})$.
- A vector $\mathbf{c}$, where $\mathbf{c}$ is random if $\mu = 1$, otherwise $\mathbf{c}^\intercal \approx \mathbf{s}^\intercal \cdot \mathbf{D}$.

To decrypt, the $i$-th party can expand $\mathbf{C}$ and $\mathbf{h}$ to recover an LWE sample approximately equal to

$$\mathbf{s}^\intercal \mathbf{x}_i = \mathbf{s}^\intercal \cdot \left[\begin{array}{c|c} \mathbf{Q}_i & \mathbf{D} \end{array}\right] \cdot \mathbf{u}_i = \mathbf{s}^\intercal \cdot \mathbf{D} \cdot \left[\begin{array}{c|c} \mathbf{R}_i & \mathbf{I}_m \end{array}\right] \cdot \mathbf{u}_i.$$

The decryptor can therefore compare this against $\mathbf{c}^\intercal \cdot \left[\begin{array}{c|c} \mathbf{R}_i & \mathbf{I}_m \end{array}\right] \cdot \mathbf{u}_i$. If the two quantities are close, it can deduce that $\mu = 0$, otherwise $\mu = 1$.

For security we need to argue that, if all the recipients are honest, the ciphertext is pseudorandom. We can do so by appealing to the security of the obfuscated lattice preimage sampler. In short, the obfuscator will ensure that, if all $\mathbf{Q}_i$ are random, $\mathsf{Hash}(\mathbf{X})$ is a matrix for which we can securely give LWE samples. Then, we appeal to the decomposed LWE assumption. We refer the reader to the technical sections for more details.

# 2 Preliminaries

We denote the security parameter by $\lambda$. We say that a function $\mu : \mathbb{N} \to \mathbb{R}$ is negligible if $|\mu(\lambda)| = 2^{-\omega(\log \lambda)}$. We denote by $x \xleftarrow{\$} X$ sampling from a distribution $X$. We denote by $X \approx_c Y$ or $X \approx_s Y$ if the two distribution ensembles $X$ and $Y$ are computationally or statistically indistinguishable, respectively.

We use $[n]$ to denote the set $\{0, 1, \ldots, n-1\}$ and by convention all indices start counting from 0. Given any binary string $\alpha \in \{0,1\}^*$, we denote its length by $|\alpha|$ and we denote the $i$-th bit of $\alpha$ by $\alpha_i$; if this is not well-defined (either because $i < 0$ or because $i \geq |\alpha|$), we adopt the convention that $\alpha_i = 1$.

## 2.1 Linear Algebra

We denote vectors using lowercase bold font, e.g., $\mathbf{v}$. Matrices will be denoted using uppercase bold font, e.g., $\mathbf{A}$. We adopt the standard linear algebra convention in which vectors correspond to columns. We use the transposition symbol $^\intercal$ to denote row vectors. We define the Kronecker product between two matrices $\mathbf{A}$ and $\mathbf{B}$ as

$$\mathbf{A} \otimes \mathbf{B} := \begin{bmatrix} a_{0,0} \cdot \mathbf{B} & \ldots & a_{0,m-1} \cdot \mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{n-1,0} \cdot \mathbf{B} & \ldots & a_{n-1,m-1} \cdot \mathbf{B} \end{bmatrix}$$

where $n$ and $m$ respectively denote the number of rows and columns of $\mathbf{A}$ and $a_{i,j}$ denotes the entry in the $i$-th row and $j$-th column of $\mathbf{A}$. We recall that for any matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$ where the number of columns of $\mathbf{A}$ and $\mathbf{B}$ respectively match the number of rows of $\mathbf{C}$ and $\mathbf{D}$, it holds that

$$(\mathbf{A} \otimes \mathbf{B}) \cdot (\mathbf{C} \otimes \mathbf{D}) = (\mathbf{A} \cdot \mathbf{C}) \otimes (\mathbf{B} \cdot \mathbf{D}). \tag{9}$$

We use $\mathsf{vec}(\mathbf{A})$ to denote the vectorization of the matrix $\mathbf{A}$. Specifically, if $\mathbf{A} \in \mathbb{Z}^{n\times m}$, $\mathsf{vec}(\mathbf{A})$ is the $nm$-dimensional vector obtained by vertically concatenating the columns of $\mathbf{A}$. We denote the $n \times n$ identity matrix by $\mathbf{I}_n$. We recall that, for any matrix $\mathbf{A}$ with $k$ rows and $\ell$ columns, and any matrix $\mathbf{B}$ with $\ell$ rows and $m$ columns, it holds that

$$\mathsf{vec}(\mathbf{AB}) = (\mathbf{I}_m \otimes \mathbf{A})\mathsf{vec}(\mathbf{B}) = (\mathbf{B}^\intercal \otimes \mathbf{I}_k)\mathsf{vec}(\mathbf{A}). \tag{10}$$

We denote by $\|\mathbf{v}\|_\infty$ the $\ell_\infty$-norm of a vector $\mathbf{v}$ and for a matrix $\mathbf{A}$ we define $\|\mathbf{A}\|_\infty = \|\mathsf{vec}(\mathbf{A})\|_\infty$. For a matrix $\mathbf{A}$ over $\mathbb{Z}_q$, identified with its centered integer representative, we define

$$\|\mathbf{A}\|_{\infty,\infty} := \max_{\|\mathbf{v}\|_\infty=1} \|\mathbf{Av}\|_\infty = \max_i \sum_j |\mathbf{A}_{i,j}|.$$

This norm is submultiplicative, and

$$\|\mathbf{A} \otimes \mathbf{B}\|_{\infty,\infty} \le \|\mathbf{A}\|_{\infty,\infty}\|\mathbf{B}\|_{\infty,\infty}.$$

For every row vector $\mathbf{v}^\intercal$ and matrix $\mathbf{A}$ of compatible dimensions,

$$\|\mathbf{v}^\intercal\mathbf{A}\|_\infty \le \|\mathbf{v}\|_\infty\|\mathbf{A}^\intercal\|_{\infty,\infty}.$$

Moreover,

$$\left\|\begin{bmatrix}\mathbf{A} & \mathbf{B}\end{bmatrix}\right\|_{\infty,\infty} \le \|\mathbf{A}\|_{\infty,\infty} + \|\mathbf{B}\|_{\infty,\infty}.$$

We assume for simplicity that $q$ is a power of two. We define the gadget matrix $\mathbf{G} = \mathbf{I}_k \otimes \mathbf{g}^\intercal \in \mathbb{Z}_q^{k\times n}$, where $n = k \log q$ and

$$\mathbf{g}^\intercal = \begin{bmatrix}1 & 2 & \cdots & 2^{\log q-1}\end{bmatrix}.$$

As customary, $\mathbf{G}^{-1}$ denotes the binary decomposition map, applied column by column, such that $\mathbf{G}\mathbf{G}^{-1}(\mathbf{X}) = \mathbf{X}$ for every matrix $\mathbf{X}$ over $\mathbb{Z}_q$. In particular, for every $c \in \mathbb{Z}_q$,

$$\|\mathbf{G}^{-1}(c\mathbf{G})\|_{\infty,\infty} \le n.$$

The following is a straightforward consequence of the leftover hash lemma [ILL89].

**Lemma 1** (Leftover Hash Lemma)**.** *Let $m \ge 2n \log q$ and $n \log q = \omega(\log \lambda)$, then the following distributions are statistically close*

$$\left\{\mathbf{A}, \mathbf{Ax} : \mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}, \mathbf{x} \xleftarrow{\$} \mathbb{Z}_2^m\right\} \approx_s \left\{\mathbf{A}, \mathbf{y} : \mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}, \mathbf{y} \xleftarrow{\$} \mathbb{Z}_q^n\right\}.$$

## 2.2 Matrix Branching Programs

A *read-once matrix branching program* $\mathsf{BP}$ of length $d \ge 1$ is a collection

$$\mathsf{BP} = \{\mathbf{M}_{i,b} \in \mathbb{Z}_q^{w_i\times w_{i+1}}\}_{i\in[d],\, b\in\{0,1\}}$$

that evaluates, for $\alpha \in \{0,1\}^d$, to

$$\mathsf{BP}(\alpha) := \prod_{i\in[d]} \mathbf{M}_{i,\alpha_i},$$

where products are always taken in increasing order of the index. We define

$$\dim(\mathsf{BP}) := \max_{i\in[d+1]} w_i, \qquad \mathrm{norm}(\mathsf{BP}) := \max_{i\in[d],\, b\in\{0,1\}} \|\mathbf{M}_{i,b}\|_{\infty,\infty}.$$

We call the branching program *scalar-output* if $w_0 = w_d = 1$. We use the following elementary closure properties.

**Lemma 2** (Closure Under Addition)**.** *Let* $\mathsf{BP}_0 = \{\mathbf{M}_{i,b} \in \mathbb{Z}_q^{w_i \times w_{i+1}}\}_{i,b}$ *and* $\mathsf{BP}_1 = \{\mathbf{N}_{i,b} \in \mathbb{Z}_q^{v_i \times v_{i+1}}\}_{i,b}$ *be read-once matrix branching programs of the same length, with* $w_0 = v_0$ *and* $w_d = v_d$*. There is an efficiently computable read-once matrix branching program* $\mathsf{BP}_2$ *such that, for every* $\alpha \in \{0,1\}^d$*,*

$$\mathsf{BP}_2(\alpha) = \mathsf{BP}_0(\alpha) + \mathsf{BP}_1(\alpha),$$

*and*

$$\dim(\mathsf{BP}_2) \le \dim(\mathsf{BP}_0) + \dim(\mathsf{BP}_1), \qquad \operatorname{norm}(\mathsf{BP}_2) \le \operatorname{norm}(\mathsf{BP}_0) + \operatorname{norm}(\mathsf{BP}_1).$$

*Proof.* For $d = 1$, take $\mathbf{S}_{0,b} = \mathbf{M}_{0,b} + \mathbf{N}_{0,b}$. For $d \ge 2$, define

$$\mathbf{S}_{0,b} := \begin{bmatrix}\mathbf{M}_{0,b} & \mathbf{N}_{0,b}\end{bmatrix}, \qquad \mathbf{S}_{i,b} := \begin{bmatrix}\mathbf{M}_{i,b} & \mathbf{0} \\ \mathbf{0} & \mathbf{N}_{i,b}\end{bmatrix} \quad (1 \le i \le d-2),$$

and

$$\mathbf{S}_{d-1,b} := \begin{bmatrix}\mathbf{M}_{d-1,b} \\ \mathbf{N}_{d-1,b}\end{bmatrix}.$$

Multiplication of the block matrices gives the claimed function. The dimension and norm bounds follow immediately from the definitions. □

**Lemma 3** (Closure Under Multiplication)**.** *Let* $\mathsf{BP}_0$ *and* $\mathsf{BP}_1$ *be read-once matrix branching programs of the same length whose outputs lie in* $\mathbb{Z}_q^{n \times n}$*. There is an efficiently computable read-once matrix branching program* $\mathsf{BP}_2$ *such that, for every* $\alpha \in \{0,1\}^d$*,*

$$\mathsf{BP}_2(\alpha) = \mathsf{BP}_0(\alpha)\mathsf{BP}_1(\alpha),$$

*and*

$$\dim(\mathsf{BP}_2) \le n\,\dim(\mathsf{BP}_0)\dim(\mathsf{BP}_1), \qquad \operatorname{norm}(\mathsf{BP}_2) \le n\operatorname{norm}(\mathsf{BP}_0)\operatorname{norm}(\mathsf{BP}_1).$$

*Proof.* For each $r \in [n]$, let

$$\mathbf{U}_r := \mathbf{I}_n \otimes \mathbf{e}_r^\intercal, \qquad \mathbf{V}_r := \mathbf{e}_r \otimes \mathbf{I}_n,$$

where $\mathbf{e}_r$ is the $r$-th standard basis vector of $\mathbb{Z}_q^n$. For all $\mathbf{X}, \mathbf{Y} \in \mathbb{Z}_q^{n \times n}$,

$$\mathbf{XY} = \sum_{r \in [n]} \mathbf{U}_r(\mathbf{X} \otimes \mathbf{Y})\mathbf{V}_r.$$

The transition matrices $\mathbf{M}_{i,b} \otimes \mathbf{N}_{i,b}$ compute $\mathsf{BP}_0(\alpha) \otimes \mathsf{BP}_1(\alpha)$. For each $r$, absorb $\mathbf{U}_r$ and $\mathbf{V}_r$ into the first and last transitions, respectively, and add the resulting $n$ branching programs using Lemma 2. Submultiplicativity of the Kronecker-product norm gives the stated bounds. □

### 2.3 Discrete Gaussians and Lattices

Let $\mathcal{D}_\sigma$ denote the discrete Gaussian distribution with parameter $\sigma$. We recall the following standard fact about Gaussians.

**Lemma 4** (Gaussian Flooding)**.** *If* $X$ *is sampled from* $\mathcal{D}_\sigma$ *then for any fixed integer* $Y$ *such that* $|Y| \le \tau$ *and* $\sigma/\tau = 2^{\omega(\log \lambda)}$*, the following distributions are statistically close:*

$$X \approx_s X + Y$$

*where* $X \leftarrow \mathcal{D}_\sigma$*.*

For $\sigma = \omega(\sqrt{\log(\lambda)})$, we have the following tail bound on the discrete Gaussian, which follows as an immediate consequence of [Ban93]:

$$\Pr\left[|x| \geq \sqrt{\lambda} \cdot \sigma : \ x \xleftarrow{\$} \mathcal{D}_\sigma\right] = O(2^{-\lambda}). \tag{11}$$

**Preimage Sampling.** We recall the structure and properties of the Micciancio–Peikert preimage sampling algorithm [MP12]. A preimage of a vector $\mathbf{x} \in \mathbb{Z}^n$ under a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ is a low-norm vector $\mathbf{y} \in \mathbb{Z}^m$ such that $\mathbf{A}\mathbf{y} = \mathbf{x}$. Given a trapdoor $\mathbf{T}$, i.e., a low-norm matrix such that $\mathbf{A}\mathbf{T} = \mathbf{G}$, one can easily find a preimage by setting $\mathbf{y} = \mathbf{T}\mathbf{G}^{-1}(\mathbf{x})$. However, such a preimage leaks information about $\mathbf{T}$, so the goal of a preimage sampler is to generate $\mathbf{y}$ according to some distribution that is independent of $\mathbf{T}$.

In this work we are going to use a simplified version of the algorithm described in [MP12], which we refer to as ShiftedPreimage. Given an initial low-norm vector $\mathbf{y}'$, the sampler adds a low-norm offset to it to get a preimage of $\mathbf{x}$. More precisely, the output of the algorithm is defined as

$$\mathsf{ShiftedPreimage}(\mathbf{A}, \mathbf{T}, \mathbf{x}, \mathbf{y}') \mapsto \mathbf{y}' - \mathbf{T}\mathbf{G}^{-1}(\mathbf{A}\mathbf{y}' - \mathbf{x}).$$

Using ShiftedPreimage, one immediately obtains a very simple preimage sampler based on noise flooding: Pick some bound $p \ll q$ and sample

$$\mathbf{y}' \xleftarrow{\$} \left[-\frac{p-1}{2}, \frac{p-1}{2}\right]^m$$

viewed as a subset of $\mathbb{Z}_q^m$, then run ShiftedPreimage. If $p \geq 2^{\omega(\log(\lambda))}\|\mathbf{T}\|_\infty$ then the shift $\mathbf{T}\mathbf{G}^{-1}(\mathbf{t})$ applied to $\mathbf{y}'$ affects its distribution negligibly (other than changing it to be a valid preimage), which makes it a valid preimage sampler.

We shall clarify that we are not going to use explicitly the indistinguishability property of the sampler, but it rather serves as an intuition for our construction. Later on in Section 5.2 we will only use the definition of ShiftedPreimage and security will only appear as part of a more complex argument.

**Learning with Errors.** The fundamental problem underlying lattice-based cryptography is the *learning with errors* (LWE) problem [Reg05] which, roughly speaking, postulates that it is hard to distinguish between random vectors and random noisy linear equations. It is known [Reg05] that, for appropriate parameters, the hardness of the LWE problem follows from the worst-case quantum hardness of SIVP and classical hardness of GapSVP. Below we recall the *decomposed LWE* problem, a recently introduced variant of the LWE problem [AMR25a].

**Definition 1** (Decomposed LWE)**.** *Let $k = k(\lambda)$, $m = m(\lambda)$, $\sigma = \sigma(\lambda)$, $\overline{\sigma} = \overline{\sigma}(\lambda)$, $t = t(\lambda)$, and $q = q(\lambda)$ be integer parameters. Define $n = k \cdot \log q$. The decomposed LWE assumption postulates the following distributions are computationally indistinguishable*

$$\{\mathbf{A}, \mathbf{B}, (\mathbf{I}_t \otimes \mathbf{s}^\intercal) \cdot (\mathbf{A}\mathbf{B} - \mathbf{I}_t \otimes \mathbf{G}) + \mathbf{E}\} \approx_c \{\mathbf{A}, \mathbf{B}, \mathbf{U}\}$$

*where $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^k$, $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{t \cdot k \times m}$, $\mathbf{B} \xleftarrow{\$} \mathcal{D}_\sigma^{m \times t \cdot n}$, $\mathbf{E} \xleftarrow{\$} \mathcal{D}_{\overline{\sigma}}^{t \times t \cdot n}$, and $\mathbf{U} \xleftarrow{\$} \mathbb{Z}_q^{t \times t \cdot n}$.*

By the symmetry of the Gaussian distribution, the formulation with $\mathbf{AB} + \mathbf{I}_t \otimes \mathbf{G}$ is equivalent. In [AMR25a] it is shown that the decomposed LWE assumption remains hard (with a reduction to Definition 1) even if the distinguisher is additionally given LWE samples with the same secret with respect to an auxiliary random matrix. We omit a formal definition since its hardness follows by Definition 1.

### 2.4 Homomorphic Encodings

We recall the syntax of homomorphic encodings from [AMR25b, AMR25a, Wee25], which were inspired by the work of Wee [Wee24]. In fact, we define here a simplified (non recursive) version that is going to be sufficient for this work.

**Parameters.** For convenience, we summarize here the parameters involved in the construction:

- The modulus $q = q(\lambda)$.
- The lattice dimension $k = k(\lambda)$ and $n = k \log q$.
- The matrix dimension $m = m(\lambda)$.
- The compression parameter $t = t(\lambda)$.
- The gaussian parameter $\sigma = \sigma(\lambda)$.

**Syntax.** The public interface consists of the following algorithms.

- $\mathsf{Setup}(1^\lambda, 1^t)$: On input the security parameter $1^\lambda$ and the compression parameter $1^t$, the setup algorithm samples

$$\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{(t\cdot k)\times m} \qquad \text{and} \qquad \mathbf{B} \xleftarrow{\$} \mathcal{D}_\sigma^{m\times(t\cdot n)}.$$

  The algorithm returns the public parameters $\mathsf{pk} = (\mathbf{A}, \mathbf{B})$.

  For convenience, we denote by $\mathbf{Z} = \left[\ \mathbf{Z}_0 \ \middle|\ \dots \ \middle|\ \mathbf{Z}_{t-1}\ \right] \in \mathbb{Z}_q^{k\times nt^2}$, where $\mathbf{Z}_i \in \mathbb{Z}_q^{k\times nt}$ is the $i$-th block of $k$ rows of the matrix $\mathbf{AB} - \mathbf{I}_t \otimes \mathbf{G}$. Note that $\mathbf{Z}$ is publicly computable from the public parameters, so we can assume without loss of generality that it is contained in $\mathsf{pk}$. Up to a fixed reordering of coordinates, $\mathbf{s}^\intercal\mathbf{Z}$ is the vectorization of $(\mathbf{I}_t \otimes \mathbf{s}^\intercal)(\mathbf{AB} - \mathbf{I}_t \otimes \mathbf{G})$.

- $\mathsf{Hash}(\mathsf{pk}, \mathbf{x})$: On input the public parameters $\mathsf{pk}$ and a vector $\mathbf{x} \in \mathbb{Z}_q^t$, the hash function returns

$$\mathsf{Hash}(\mathsf{pk}, \mathbf{x}) := \mathsf{vec}(\mathbf{B}(\mathbf{x} \otimes \mathbf{I}_n)) \in \mathbb{Z}_q^{mn}.$$

  In particular, the hash function is deterministic and linear over $\mathbb{Z}_q$. We denote the matrix of this linear map by $\mathbf{D} \in \mathbb{Z}_q^{mn\times t}$, so that $\mathsf{Hash}(\mathsf{pk}, \mathbf{x}) = \mathbf{Dx}$. Except with negligible probability over $\mathsf{pk}$,

$$\|\mathbf{D}\|_{\infty,\infty} \le t\sigma\sqrt{\lambda}$$

**Encoding Expansion.** It is shown in [AMR25a] that it is possible to publicly expand the encoding of the hash of a vector into the encoding of the vector itself. We summarize this fact via the following statement.

**Lemma 5** (Encoding Expansion)**.** *There exists a pair of efficient deterministic algorithms:*

- $\mathsf{ExpandM}(\mathsf{pk}) \mapsto \mathbf{P} \in \mathbb{Z}_q^{(m\cdot n^2)\times(t\cdot n)}$,
- $\mathsf{ExpandC}(\mathsf{pk}, \mathbf{x}) \mapsto \mathbf{Q}_{\mathbf{x}} \in \mathbb{Z}_q^{(n\cdot t^2)\times(t\cdot n)}$.

*such that for all* $\mathsf{pk}$ *in the support of* $\mathsf{Setup}(1^\lambda, 1^t)$*, all matrices* $\mathbf{M} \in \mathbb{Z}_q^{n\times(m\cdot n^2)}$ *and all vectors* $\mathbf{x} \in \mathbb{Z}_q^t$ *it holds that*

$$\left[\ \mathbf{G}\cdot(\mathbf{M} - \mathsf{Hash}(\mathsf{pk},\mathbf{x})^\intercal \otimes \mathbf{I}_n) \ \middle|\ \mathbf{Z}\ \right] \cdot \begin{bmatrix}\mathbf{P}\\ \mathbf{Q}_{\mathbf{x}}\end{bmatrix} = \mathbf{G}\cdot(\mathbf{M}\cdot\mathbf{P} - \mathbf{x}^\intercal \otimes \mathbf{I}_n)$$

*Furthermore,*

$$\|\mathbf{P}^\intercal\|_{\infty,\infty} \le nm, \qquad \|\mathbf{P}\|_{\infty,\infty} \le t, \qquad \|\mathbf{Q}_{\mathbf{x}}\|_{\infty,\infty} \le \|\mathbf{x}\|_\infty.$$

*Moreover,* $\mathbf{P}$ *depends only on* $\mathsf{pk}$ *and may be precomputed as part of the public parameters. We use the standard parallel implementation of* $\mathsf{ExpandC}$*, whose Boolean circuit depth is* $O\left(\log(mnt) + \log\log q\right)$.

Setting $\mathbf{M} = \mathbf{0}$ in the expansion equation and multiplying by $-1$ gives the equivalent identity

$$\mathbf{G}(\mathsf{Hash}(\mathsf{pk},\mathbf{x})^\intercal \otimes \mathbf{I}_n)\mathbf{P} - \mathbf{Z}\mathbf{Q}_{\mathbf{x}} = \mathbf{G}(\mathbf{x}^\intercal \otimes \mathbf{I}_n). \tag{12}$$

Whenever $t$ is even, we write $\mathbf{P} = [\mathbf{P}_0 \mid \mathbf{P}_1]$ and $\mathbf{Q}_{\mathbf{x}} = [\mathbf{Q}_{\mathbf{x},0} \mid \mathbf{Q}_{\mathbf{x},1}]$ for their equal column blocks.

# 3 Tree Encodings

In this section we describe tree encodings through read-once matrix branching programs. We use the homomorphic encodings of Section 2.4 with $t = 2mn$.

Accordingly, $\mathbf{D} = [\mathbf{D}_0 \mid \mathbf{D}_1]$, where $\mathbf{D}_0, \mathbf{D}_1 \in \mathbb{Z}_q^{mn\times mn}$, and we write

$$\mathbf{P} = [\mathbf{P}_0 \mid \mathbf{P}_1], \qquad \mathbf{Q}_{\mathbf{x}} = [\mathbf{Q}_{\mathbf{x},0} \mid \mathbf{Q}_{\mathbf{x},1}],$$

where $\mathbf{P}_b \in \mathbb{Z}_q^{mn^2\times mn^2}$ and $\mathbf{Q}_{\mathbf{x},b} \in \mathbb{Z}_q^{nt^2\times mn^2}$. The setup algorithm used throughout the paper is the following wrapper, with $\mathbf{P} = \mathsf{ExpandM}(\mathsf{pk})$ precomputed and included in the public parameters:

$$\mathsf{TreeSetup}(1^\lambda): \quad \mathsf{pk}_0 \xleftarrow{\$} \mathsf{Setup}(1^\lambda, 1^t), \quad \mathbf{P} \leftarrow \mathsf{ExpandM}(\mathsf{pk}_0), \quad \mathsf{pk} := (\mathsf{pk}_0, \mathbf{P}).$$

All scalar absolute values below refer to centered representatives in $\mathbb{Z}_q$.

### 3.1 Hashing Branching Programs

We first recall the Merkle-tree extension of $\mathsf{Hash}$. For leaves $\{\mathbf{x}_\alpha \in \mathbb{Z}_q^{mn}\}_{\alpha\in\{0,1\}^d}$ and every $\rho \in \{0,1\}^{<d}$, define

$$\mathbf{h}_\alpha := \mathbf{x}_\alpha, \qquad \mathbf{y}_\rho := \begin{bmatrix}\mathbf{h}_{\rho 0}\\ \mathbf{h}_{\rho 1}\end{bmatrix}, \qquad \mathbf{h}_\rho := \mathsf{Hash}(\mathsf{pk}, \mathbf{y}_\rho) = \mathbf{D}\mathbf{y}_\rho.$$

We set

$$\mathsf{MerkleHash}(\mathsf{pk}, \{\mathbf{x}_\alpha\}_\alpha) := \mathbf{h}_\emptyset.$$

By Equation (12), for every $\rho \in \{0,1\}^{<d}$ and $b \in \{0,1\}$,

$$\mathbf{G}(\mathbf{h}_\rho^\intercal \otimes \mathbf{I}_n)\mathbf{P}_b - \mathbf{Z}\mathbf{Q}_{\mathbf{y}_\rho, b} = \mathbf{G}(\mathbf{h}_{\rho b}^\intercal \otimes \mathbf{I}_n). \tag{13}$$

Let $\mathbf{e}_0$ denote the first standard basis vector of $\mathbb{Z}_q^{mn}$.

The first observation is that the Merkle root of an exponentially large truth table represented by a branching program is efficiently computable.

**Lemma 6** (Hashing a Branching Program)**.** *Let $\mathsf{BP} = \{\mathbf{M}_{i,b}\}_{i\in[d],b\in\{0,1\}}$ be a scalar-output read-once matrix branching program. There is a deterministic algorithm that computes*

$$\mathsf{MerkleHash}\left(\mathsf{pk}, \{\mathsf{BP}(\beta)\mathbf{e}_0\}_{\beta\in\{0,1\}^d}\right)$$

*in time $\mathsf{poly}(n, m, \log q, d, \dim(\mathsf{BP}))$.*

*Proof.* Linearity of the hash gives

$$\begin{aligned}\mathsf{MerkleHash}\left(\mathsf{pk}, \{\mathsf{BP}(\beta)\mathbf{e}_0\}_\beta\right) &= \sum_{\beta\in\{0,1\}^d} \left(\prod_{i\in[d]} \mathbf{D}_{\beta_i}\right) \mathbf{e}_0 \mathsf{BP}(\beta)\\ &= \left(\prod_{i\in[d]} (\mathbf{D}_0 \otimes \mathbf{M}_{i,0} + \mathbf{D}_1 \otimes \mathbf{M}_{i,1})\right)\mathbf{e}_0.\end{aligned}$$

The matrices in the last expression have polynomial dimension and can therefore be multiplied in polynomial time. □

The same argument computes every node on a requested root-to-leaf path. Indeed, if $\rho \in \{0,1\}^j$, the leaves below $\rho$ are represented by the scalar-output branching program

$$\mathsf{BP}_\rho(\gamma) := \left(\prod_{i=0}^{j-1} \mathbf{M}_{i,\rho_i}\right)\left(\prod_{i=j}^{d-1} \mathbf{M}_{i,\gamma_{i-j}}\right).$$

We next combine this efficient hashing procedure with encoding expansion.

**Lemma 7** (Branching-Program Expansion)**.** *Let $\mathsf{BP}$ be a scalar-output read-once matrix branching program. For every $\alpha \in \{0,1\}^d$, there is a matrix $\mathbf{T}_{\mathsf{BP},\alpha} \in \mathbb{Z}_q^{nt^2\times n}$ such that*

$$\mathbf{G}\left(\mathsf{MerkleHash}\left(\mathsf{pk}, \{\mathsf{BP}(\beta)\mathbf{e}_0\}_\beta\right)^\intercal \otimes \mathbf{I}_n\right)\left(\prod_{i\in[d]} \mathbf{P}_{\alpha_i}\right)(\mathbf{e}_0 \otimes \mathbf{I}_n) + \mathbf{Z}\mathbf{T}_{\mathsf{BP},\alpha} = \mathsf{BP}(\alpha)\mathbf{G}. \tag{14}$$

*Moreover,* $\mathbf{T}_{\mathsf{BP},\alpha}$ *is computable in time* $\mathsf{poly}(n, m, \log q, d, \dim(\mathsf{BP}))$ *and*

$$\|\mathbf{T}_{\mathsf{BP},\alpha}\|_{\infty,\infty} \le \max_{\beta\in\{0,1\}^d} |\mathsf{BP}(\beta)| \sum_{r=0}^{d-1} (t\|\mathbf{D}\|_{\infty,\infty})^r . \tag{15}$$

*Proof.* Fix $\alpha \in \{0,1\}^d$ and write $\alpha_{<j} := \alpha_0 \cdots \alpha_{j-1}$, with $\alpha_{<0} = \emptyset$. Let $\mathbf{K}_0 := \mathbf{0} \in \mathbb{Z}_q^{nt^2\times mn^2}$ and, for $j \in [d]$, define

$$\mathbf{K}_{j+1} := \mathbf{K}_j\mathbf{P}_{\alpha_j} - \mathbf{Q}_{\mathbf{y}_{\alpha_{<j}},\alpha_j}.$$

We claim that, for every $j \in \{0,\dots,d\}$,

$$\begin{bmatrix}\mathbf{h}_\emptyset^\intercal \otimes \mathbf{G} & \mathbf{Z}\end{bmatrix} \begin{bmatrix}\prod_{i<j} \mathbf{P}_{\alpha_i} \\ \mathbf{K}_j\end{bmatrix} = \mathbf{h}_{\alpha_{<j}}^\intercal \otimes \mathbf{G}. \tag{16}$$

The claim is immediate for $j = 0$. If it holds for $j < d$, then

$$\begin{aligned}\begin{bmatrix}\mathbf{h}_\emptyset^\intercal \otimes \mathbf{G} & \mathbf{Z}\end{bmatrix} \begin{bmatrix}\prod_{i<j+1} \mathbf{P}_{\alpha_i} \\ \mathbf{K}_{j+1}\end{bmatrix} &= \left(\begin{bmatrix}\mathbf{h}_\emptyset^\intercal \otimes \mathbf{G} & \mathbf{Z}\end{bmatrix} \begin{bmatrix}\prod_{i<j} \mathbf{P}_{\alpha_i} \\ \mathbf{K}_j\end{bmatrix}\right) \mathbf{P}_{\alpha_j} - \mathbf{Z}\mathbf{Q}_{\mathbf{y}_{\alpha_{<j}},\alpha_j} \\ &= (\mathbf{h}_{\alpha_{<j}}^\intercal \otimes \mathbf{G})\mathbf{P}_{\alpha_j} - \mathbf{Z}\mathbf{Q}_{\mathbf{y}_{\alpha_{<j}},\alpha_j} \\ &= \mathbf{h}_{\alpha_{<j+1}}^\intercal \otimes \mathbf{G},\end{aligned}$$

where the last equality is Equation (13). Thus Equation (16) follows by induction.

Set

$$\mathbf{T}_{\mathsf{BP},\alpha} := \mathbf{K}_d(\mathbf{e}_0 \otimes \mathbf{I}_n).$$

Applying Equation (16) for $j = d$ and using $\mathbf{h}_\alpha = \mathsf{BP}(\alpha)\mathbf{e}_0$ gives Equation (14).

Each vector $\mathbf{y}_{\alpha_{<j}}$ is efficiently computable by Lemma 6. Furthermore,

$$\|\mathbf{y}_{\alpha_{<j}}\|_\infty \le \|\mathbf{D}\|_{\infty,\infty}^{d-j-1} \max_{\beta\in\{0,1\}^d} |\mathsf{BP}(\beta)|.$$

Consequently, Lemma 5 and submultiplicativity imply

$$\|\mathbf{K}_{j+1}\|_{\infty,\infty} \le t\|\mathbf{K}_j\|_{\infty,\infty} + \|\mathbf{D}\|_{\infty,\infty}^{d-j-1} \max_\beta |\mathsf{BP}(\beta)|.$$

Unrolling this recurrence and using $\|\mathbf{e}_0 \otimes \mathbf{I}_n\|_{\infty,\infty} = 1$ proves Equation (15). □

### 3.2 Homomorphic Operations

For a function $f : \{0,1\}^d \to \mathbb{Z}_q$, we maintain an efficiently computable read-once matrix branching program $\mathsf{MBP}_f$ with output in $\mathbb{Z}_q^{n\times n}$ and, for every $\alpha \in \{0,1\}^d$, an efficiently computable matrix $\mathbf{H}_{f,\alpha} \in \mathbb{Z}_q^{nt^2\times n}$ such that

$$\mathbf{G}\mathsf{MBP}_f(\alpha) - \mathbf{Z}\mathbf{H}_{f,\alpha} = f(\alpha)\mathbf{G}. \tag{17}$$

We first establish the invariant for input and constant wires. For $i \in [d]$, let $\mathsf{BP}_i$ be the scalar-output branching program whose transition matrices are all $[1]$, except that $\mathbf{M}_{i,0} = [0]$. Thus $\mathsf{BP}_i(\alpha) = \alpha_i$. Let $\mathsf{BP}_\perp$ have every transition equal to $[1]$, so that $\mathsf{BP}_\perp(\alpha) = 1$. For $u \in [d] \cup \{\perp\}$, define

$$\mathbf{h}_u := \mathsf{MerkleHash}\,(\mathsf{pk}, \{\mathsf{BP}_u(\beta)\mathbf{e}_0\}_\beta)$$

and

$$\mathbf{M}_u := \mathbf{G}^{-1}\left(\mathbf{G}(\mathbf{h}_u^\intercal \otimes \mathbf{I}_n)\right) \in \mathbb{Z}_q^{n\times mn^2}.$$

Finally, set

$$\mathsf{MBP}_{\Pi_i}(\alpha) := \mathbf{M}_i \left(\prod_{j\in[d]} \mathbf{P}_{\alpha_j}\right)(\mathbf{e}_0 \otimes \mathbf{I}_n), \qquad \mathbf{H}_{\Pi_i,\alpha} := -\mathbf{T}_{\mathsf{BP}_i,\alpha},$$

and

$$\mathsf{MBP}_1(\alpha) := \mathbf{M}_\perp \left(\prod_{j\in[d]} \mathbf{P}_{\alpha_j}\right)(\mathbf{e}_0 \otimes \mathbf{I}_n), \qquad \mathbf{H}_{1,\alpha} := -\mathbf{T}_{\mathsf{BP}_\perp,\alpha}.$$

The fixed matrices on the left and right are absorbed into the first and last transitions, respectively. By Lemma 7, $(\Pi_i, \mathsf{MBP}_{\Pi_i}, \mathbf{H}_{\Pi_i,\alpha})$ and $(1, \mathsf{MBP}_1, \mathbf{H}_{1,\alpha})$ satisfy Equation (17). Moreover,

$$\dim(\mathsf{MBP}_{\Pi_i}), \dim(\mathsf{MBP}_1) \le mn^2, \qquad \mathrm{norm}(\mathsf{MBP}_{\Pi_i}), \mathrm{norm}(\mathsf{MBP}_1) \le tmn^2,$$

and

$$\|\mathbf{H}_{\Pi_i,\alpha}\|_{\infty,\infty}, \|\mathbf{H}_{1,\alpha}\|_{\infty,\infty} \le \sum_{r=0}^{d-1} \left(t\|\mathbf{D}\|_{\infty,\infty}\right)^r. \tag{18}$$

The invariant is preserved by the arithmetic operations used below.

**Lemma 8** (Addition)**.** *Suppose* $(f, \mathsf{MBP}_f, \mathbf{H}_{f,\alpha})$ *and* $(g, \mathsf{MBP}_g, \mathbf{H}_{g,\alpha})$ *satisfy Equation* (17)*. There is an efficiently computable pair* $(\mathsf{MBP}_{f+g}, \mathbf{H}_{f+g,\alpha})$ *satisfying Equation* (17)*, with*

$$\begin{aligned}
\dim(\mathsf{MBP}_{f+g}) &\le \dim(\mathsf{MBP}_f) + \dim(\mathsf{MBP}_g),\\
\mathrm{norm}(\mathsf{MBP}_{f+g}) &\le \mathrm{norm}(\mathsf{MBP}_f) + \mathrm{norm}(\mathsf{MBP}_g),\\
\|\mathbf{H}_{f+g,\alpha}\|_{\infty,\infty} &\le \|\mathbf{H}_{f,\alpha}\|_{\infty,\infty} + \|\mathbf{H}_{g,\alpha}\|_{\infty,\infty}.
\end{aligned}$$

*Proof.* Use Lemma 2 to construct a branching program satisfying

$$\mathsf{MBP}_{f+g}(\alpha) = \mathsf{MBP}_f(\alpha) + \mathsf{MBP}_g(\alpha),$$

and set

$$\mathbf{H}_{f+g,\alpha} := \mathbf{H}_{f,\alpha} + \mathbf{H}_{g,\alpha}.$$

Adding the two instances of Equation (17) proves correctness; the norm bounds follow by subadditivity. □

**Lemma 9** (Multiplication by a Constant)**.** *Suppose* $(f, \mathsf{MBP}_f, \mathbf{H}_{f,\alpha})$ *satisfies Equation* (17)*, and let* $c \in \mathbb{Z}_q$*. There is an efficiently computable pair* $(\mathsf{MBP}_{cf}, \mathbf{H}_{cf,\alpha})$ *satisfying Equation* (17)*, with*

$$\begin{aligned}
\dim(\mathsf{MBP}_{cf}) &= \dim(\mathsf{MBP}_f),\\
\mathrm{norm}(\mathsf{MBP}_{cf}) &\le n\,\mathrm{norm}(\mathsf{MBP}_f),\\
\|\mathbf{H}_{cf,\alpha}\|_{\infty,\infty} &\le n\,\|\mathbf{H}_{f,\alpha}\|_{\infty,\infty}.
\end{aligned}$$

*Proof.* Let $\mathbf{R}_c := \mathbf{G}^{-1}(c\mathbf{G})$ and define

$$\mathsf{MBP}_{cf}(\alpha) := \mathsf{MBP}_f(\alpha)\mathbf{R}_c, \qquad \mathbf{H}_{cf,\alpha} := \mathbf{H}_{f,\alpha}\mathbf{R}_c,$$

where $\mathbf{R}_c$ is absorbed into the last transition. Then

$$\begin{aligned}\mathbf{G}\mathsf{MBP}_{cf}(\alpha) - \mathbf{Z}\mathbf{H}_{cf,\alpha} &= (\mathbf{G}\mathsf{MBP}_f(\alpha) - \mathbf{Z}\mathbf{H}_{f,\alpha})\,\mathbf{R}_c \\ &= f(\alpha)\mathbf{G}\mathbf{G}^{-1}(c\mathbf{G}) \\ &= cf(\alpha)\mathbf{G}.\end{aligned}$$

The bounds follow from $\|\mathbf{R}_c\|_{\infty,\infty} \le n$. □

**Lemma 10** (Multiplication)**.** *Suppose* $(f, \mathsf{MBP}_f, \mathbf{H}_{f,\alpha})$ *and* $(g, \mathsf{MBP}_g, \mathbf{H}_{g,\alpha})$ *satisfy Equation* (17)*. There is an efficiently computable pair* $(\mathsf{MBP}_{fg}, \mathbf{H}_{fg,\alpha})$ *satisfying Equation* (17)*, with*

$$\begin{aligned}\dim(\mathsf{MBP}_{fg}) &\le n\,\dim(\mathsf{MBP}_f)\dim(\mathsf{MBP}_g), \\ \mathrm{norm}(\mathsf{MBP}_{fg}) &\le n\,\mathrm{norm}(\mathsf{MBP}_f)\mathrm{norm}(\mathsf{MBP}_g),\end{aligned}$$

*and*

$$\|\mathbf{H}_{fg,\alpha}\|_{\infty,\infty} \le \|\mathbf{H}_{f,\alpha}\|_{\infty,\infty}\mathrm{norm}(\mathsf{MBP}_g)^d + |f(\alpha)|\|\mathbf{H}_{g,\alpha}\|_{\infty,\infty}.$$

*Proof.* By Lemma 3, one can construct a branching program satisfying

$$\mathsf{MBP}_{fg}(\alpha) = \mathsf{MBP}_f(\alpha)\mathsf{MBP}_g(\alpha)$$

and the stated dimension and transition-norm bounds. Define

$$\mathbf{H}_{fg,\alpha} := \mathbf{H}_{f,\alpha}\mathsf{MBP}_g(\alpha) + f(\alpha)\mathbf{H}_{g,\alpha}.$$

Then

$$\begin{aligned}\mathbf{G}\mathsf{MBP}_{fg}(\alpha) - \mathbf{Z}\mathbf{H}_{fg,\alpha} &= (\mathbf{G}\mathsf{MBP}_f(\alpha) - \mathbf{Z}\mathbf{H}_{f,\alpha})\,\mathsf{MBP}_g(\alpha) - f(\alpha)\mathbf{Z}\mathbf{H}_{g,\alpha} \\ &= f(\alpha)\,(\mathbf{G}\mathsf{MBP}_g(\alpha) - \mathbf{Z}\mathbf{H}_{g,\alpha}) \\ &= f(\alpha)g(\alpha)\mathbf{G}.\end{aligned}$$

Finally, $\|\mathsf{MBP}_g(\alpha)\|_{\infty,\infty} \le \mathrm{norm}(\mathsf{MBP}_g)^d$. □

### 3.3 Flattening

The preceding operations increase the width and transition norm. We now show how to reset both while preserving the key branching-program invariant.

**Lemma 11** (Flattening)**.** *Let* $\mathsf{MBP}$ *be a read-once matrix branching program with* $\mathsf{MBP}(\alpha) \in \mathbb{Z}_q^{n\times n}$*. There is an efficiently computable read-once matrix branching program* $\widehat{\mathsf{MBP}}$ *and, for every* $\alpha \in \{0,1\}^d$*, an efficiently computable matrix* $\mathbf{H}^{\mathsf{flat}}_{\mathsf{MBP},\alpha} \in \mathbb{Z}_q^{nt^2\times n}$ *such that*

$$\mathbf{G}\widehat{\mathsf{MBP}}(\alpha) - \mathbf{Z}\mathbf{H}^{\mathsf{flat}}_{\mathsf{MBP},\alpha} = \mathbf{G}\mathsf{MBP}(\alpha). \tag{19}$$

*Moreover,* $\dim(\widehat{\mathsf{MBP}}) \le mn^4$, $\mathrm{norm}(\widehat{\mathsf{MBP}}) \le tmn^4$, *and*

$$\left\|\mathbf{H}^{\mathsf{flat}}_{\mathsf{MBP},\alpha}\right\|_{\infty,\infty} \le n^2 \max_{\beta\in\{0,1\}^d} \|\mathsf{MBP}(\beta)\|_{\infty,\infty} \sum_{r=0}^{d-1} (t\|\mathbf{D}\|_{\infty,\infty})^r . \tag{20}$$

*The construction runs in time* $\mathsf{poly}(n, m, \log q, d, \dim(\mathsf{MBP}))$.

*Proof.* Write $\mathsf{MBP} = \{\mathbf{M}_{i,b}\}_{i,b}$. For every $r, s \in [n]$, let

$$\mathsf{MBP}_{r,s}(\alpha) := \mathbf{e}_r^\intercal \mathsf{MBP}(\alpha)\mathbf{e}_s$$

be the scalar-output branching program obtained by absorbing $\mathbf{e}_r^\intercal$ and $\mathbf{e}_s$ into the first and last transitions. Its dimension and transition norm do not exceed those of $\mathsf{MBP}$. Define

$$\mathbf{h}_{r,s} := \mathsf{MerkleHash}\left(\mathsf{pk}, \{\mathsf{MBP}_{r,s}(\beta)\mathbf{e}_0\}_{\beta\in\{0,1\}^d}\right).$$

By Lemma 6, all $\mathbf{h}_{r,s}$ are efficiently computable. Applying Lemma 7 gives an efficiently computable $\mathbf{T}_{r,s,\alpha} \in \mathbb{Z}_q^{nt^2\times n}$ such that

$$\mathbf{G}(\mathbf{h}_{r,s}^\intercal \otimes \mathbf{I}_n)\left(\prod_{i\in[d]} \mathbf{P}_{\alpha_i}\right)(\mathbf{e}_0 \otimes \mathbf{I}_n) + \mathbf{Z}\mathbf{T}_{r,s,\alpha} = \mathsf{MBP}_{r,s}(\alpha)\mathbf{G}. \tag{21}$$

Fix an ordering of $[n]^2$. Let $\mathbf{R} \in \{0,1\}^{mn^4\times n}$ be the vertical stack whose $(r,s)$-block is $(\mathbf{e}_0 \otimes \mathbf{I}_n)\mathbf{e}_r\mathbf{e}_s^\intercal$. Using the same ordering, define the horizontal block matrix

$$\mathbf{M}_{\mathsf{MBP}} := \mathbf{G}^{-1}\left(\mathbf{G}\left[\mathbf{h}_{r,s}^\intercal \otimes \mathbf{I}_n\right]_{r,s\in[n]}\right) \in \mathbb{Z}_q^{n\times mn^4}.$$

The normalized branching program is

$$\widehat{\mathsf{MBP}}(\alpha) := \mathbf{M}_{\mathsf{MBP}}\left(\prod_{i\in[d]} \mathbf{I}_{n^2} \otimes \mathbf{P}_{\alpha_i}\right)\mathbf{R}. \tag{22}$$

The fixed matrices $\mathbf{M}_{\mathsf{MBP}}$ and $\mathbf{R}$ are absorbed into the first and last transitions. Define

$$\mathbf{H}^{\mathsf{flat}}_{\mathsf{MBP},\alpha} := -\sum_{r,s\in[n]} \mathbf{T}_{r,s,\alpha}\mathbf{e}_r\mathbf{e}_s^\intercal.$$

Summing Equation (21) after right multiplication by $\mathbf{e}_r\mathbf{e}_s^\intercal$ gives

$$\mathbf{G}\widehat{\mathsf{MBP}}(\alpha) - \mathbf{Z}\mathbf{H}^{\mathsf{flat}}_{\mathsf{MBP},\alpha} = \sum_{r,s\in[n]} \mathsf{MBP}_{r,s}(\alpha)\mathbf{G}\mathbf{e}_r\mathbf{e}_s^\intercal = \mathbf{G}\mathsf{MBP}(\alpha),$$

which proves Equation (19). The intermediate width in Equation (22) is $mn^4$. The matrix $\mathbf{M}_{\mathsf{MBP}}$ is binary, $\|\mathbf{R}\|_{\infty,\infty} = 1$, and

$$\|\mathbf{M}_{\mathsf{MBP}}(\mathbf{I}_{n^2} \otimes \mathbf{P}_b)\|_{\infty,\infty} \le tmn^4.$$

This proves the branching-program bounds. Finally,

$$\left\|\mathbf{H}^{\mathsf{flat}}_{\mathsf{MBP},\alpha}\right\|_{\infty,\infty} \le \sum_{r,s\in[n]} \|\mathbf{T}_{r,s,\alpha}\|_{\infty,\infty} \le n^2 \max_{\beta\in\{0,1\}^d} \|\mathsf{MBP}(\beta)\|_{\infty,\infty} \sum_{j=0}^{d-1} (t\|\mathbf{D}\|_{\infty,\infty})^j$$

by Lemma 7. □

If $(f, \mathsf{MBP}_f, \mathbf{H}_{f,\alpha})$ satisfies Equation (17), then flattening and the update $\widehat{\mathbf{H}}_{f,\alpha} := \mathbf{H}_{f,\alpha} + \mathbf{H}^{\mathsf{flat}}_{\mathsf{MBP}_f,\alpha}$ give

$$\mathbf{G}\widehat{\mathsf{MBP}}_f(\alpha) - \mathbf{Z}\widehat{\mathbf{H}}_{f,\alpha} = f(\alpha)\mathbf{G}.$$

Importantly, Equation (22) gives the universal normal form

$$\widehat{\mathsf{MBP}}_f(\alpha) = \mathbf{M}_f\overline{\mathbf{P}}_\alpha, \qquad \overline{\mathbf{P}}_\alpha := \left(\prod_{i\in[d]} \mathbf{I}_{n^2} \otimes \mathbf{P}_{\alpha_i}\right)\mathbf{R}, \tag{23}$$

where $\mathbf{M}_f := \mathbf{M}_{\mathsf{MBP}_f}$ depends on $f$, while $\overline{\mathbf{P}}_\alpha$ depends only on $\mathsf{pk}$ and $\alpha$.

### 3.4 Putting Things Together

An arithmetic circuit is $B$-bounded if it consists of addition, multiplication, multiplication-by-a-constant, input, and constant gates, and the centered absolute value of every wire on every input is at most $B$.

**Theorem 5** (Key Equation)**.** *Let $f = (f_0, \ldots, f_{r-1}) : \{0,1\}^d \to \mathbb{Z}_q^r$, where every coordinate is computed by a depth-$\delta$, $B$-bounded arithmetic circuit and $B \geq 1$. There exist deterministic polynomial-time algorithms*

- $\mathsf{TreeEvalM}(\mathsf{pk}, f) \mapsto \mathbf{M}_f \in \mathbb{Z}_q^{n\times rmn^4}$,
- $\mathsf{TreeEvalH}(\mathsf{pk}, f, \alpha) \mapsto \mathbf{H}_{f,\alpha} \in \mathbb{Z}_q^{nt^2\times rn}$,
- $\mathsf{TreeExpand}(\mathsf{pk}, 1^r, \alpha) \mapsto \overline{\mathbf{P}}_\alpha^{(r)} \in \mathbb{Z}_q^{rmn^4\times rn}$,

*such that, for every $\alpha \in \{0,1\}^d$,*

$$\mathbf{G}\left(\mathbf{M}_f\overline{\mathbf{P}}_\alpha^{(r)} - f(\alpha)^\intercal \otimes \mathbf{I}_n\right) = \mathbf{Z}\mathbf{H}_{f,\alpha}. \tag{24}$$

*Moreover, $\mathbf{M}_f$ is binary and*

$$\begin{aligned}
\|\mathbf{M}_f^\intercal\|_{\infty,\infty} &\leq n,\\
\|(\overline{\mathbf{P}}_\alpha^{(r)})^\intercal\|_{\infty,\infty} &\leq n(mn)^d,\\
\|\mathbf{H}_{f,\alpha}^\intercal\|_{\infty,\infty} &\leq nt^2\left((tmn^4)^d + B\right)^\delta\left(1 + n^2\left(n(tmn^4)^2\right)^d\right)\sum_{j=0}^{d-1}(t\|\mathbf{D}\|_{\infty,\infty})^j.
\end{aligned}$$

*Finally,* $\mathsf{TreeExpand}$ *has Boolean circuit depth $O\left(\log d \cdot (\log(mn) + \log\log q)\right)$, and* $\mathsf{TreeEvalH}$ *has depth $O\left((\delta)\log d \cdot (\log(mn) + \log\log q)\right)$.*

*Proof.* We first consider a scalar-output circuit. Initialize its input and constant-one wires as in Section 3.2; every other constant wire follows from Lemma 9. Evaluate the circuit gate by gate using Lemmas 8 to 10, and apply Lemma 11 after every gate. If the output is an input or constant wire, apply flattening once to the output. Correctness follows inductively from Equation (17) and Equation (19).

Every normalized branching program has dimension at most $mn^4$ and transition norm at most $tmn^4$. Before the next normalization, addition, constant multiplication, and multiplication produce branching programs with transition norm at most $2tmn^4$, $ntmn^4$, and $n(tmn^4)^2$, respectively. Thus every pre-normalization branching program has transition norm at most $n(tmn^4)^2$, and

$$\max_{\beta\in\{0,1\}^d} \|\mathsf{MBP}(\beta)\|_{\infty,\infty} \le \left(n(tmn^4)^2\right)^d. \tag{25}$$

By Lemma 11, each flattening step therefore adds a correction of norm at most

$$n^2\left(n(tmn^4)^2\right)^d \sum_{j=0}^{d-1} (t\|\mathbf{D}\|_{\infty,\infty})^j. \tag{26}$$

It remains to bound the correction inherited from the input wires. Addition increases its norm by at most a factor of 2, multiplication by a constant by at most $n$, and multiplication by Lemma 10 by at most

$$(tmn^4)^d + B.$$

Since $(tmn^4)^d + B \ge \max\{2, n\}$, unrolling this recurrence from the initial bound Equation (18) and adding Equation (26) at every level shows that the scalar correction has norm at most

$$\left((tmn^4)^d + B\right)^\delta \left(1 + n^2\left(n(tmn^4)^2\right)^d\right) \sum_{j=0}^{d-1} (t\|\mathbf{D}\|_{\infty,\infty})^j.$$

Indeed, a depth-zero input or constant circuit is covered by the factor in parentheses, while at every subsequent layer the inherited correction is multiplied by at most $(tmn^4)^d + B$ and one more flattening correction is added. The resulting geometric sum is bounded by the displayed expression because $(tmn^4)^d + B \ge 2$.

The last normalization has the form Equation (23). Thus $\mathsf{TreeEvalM}$ returns its binary left factor $\mathbf{M}_f$, while $\mathsf{TreeExpand}$ returns

$$\overline{\mathbf{P}}_\alpha = \left(\prod_{i\in[d]} \mathbf{I}_{n^2} \otimes \mathbf{P}_{\alpha_i}\right) \mathbf{R}.$$

The transpose bounds follow because $\mathbf{M}_f$ is binary with $n$ rows, $\|\mathbf{R}^\intercal\|_{\infty,\infty} = n$, and $\|\mathbf{P}_b^\intercal\|_{\infty,\infty} \le mn$ by Lemma 5; block-diagonal repetition preserves the transpose norm of the expansion. The scalar key equation is exactly Equation (17) written in this normal form.

For vector output, run the scalar construction for every coordinate in parallel and define

$$\mathbf{M}_f := \begin{bmatrix}\mathbf{M}_{f_0} & \cdots & \mathbf{M}_{f_{r-1}}\end{bmatrix}, \qquad \overline{\mathbf{P}}_\alpha^{(r)} := \mathbf{I}_r \otimes \overline{\mathbf{P}}_\alpha, \qquad \mathbf{H}_{f,\alpha} := \begin{bmatrix}\mathbf{H}_{f_0,\alpha} & \cdots & \mathbf{H}_{f_{r-1},\alpha}\end{bmatrix}.$$

Concatenating the scalar equations gives Equation (24). Horizontal concatenation multiplies the row-sum bounds by at most $r$. It does not increase the maximum column sum beyond the maximum among the scalar blocks; since a scalar correction has $nt^2$ rows,

$$\|\mathbf{H}_{f,\alpha}^\intercal\|_{\infty,\infty} \le nt^2\left((tmn^4)^d + B\right)^\delta \left(1 + n^2\left(n(tmn^4)^2\right)^d\right) \sum_{j=0}^{d-1} (t\|\mathbf{D}\|_{\infty,\infty})^j.$$

For the depth bounds, $\mathbf{P}$ is precomputed by $\mathsf{TreeSetup}$, and we use the parallel implementation of $\mathsf{ExpandC}$ from Lemma 5. Select the matrices indexed by the bits of $\alpha$ and multiply them in a balanced tree. This gives the stated depth for $\mathsf{TreeExpand}$. For Lemma 7, expand the recurrence for $\mathbf{K}_d$ as a sum of the $d$ matrices $\mathbf{Q}_{\mathbf{y}_{\alpha_{<j}},\alpha_j}$ multiplied by suffix products of the $\mathbf{P}_{\alpha_i}$. All suffix products, subtree hashes, and the final sum can be computed by balanced trees in the same asymptotic depth. The homomorphic correction is then computed one circuit layer at a time, and the coordinates are processed in parallel. This proves the depth bound for $\mathsf{TreeEvalH}$. □

From the gate-by-gate proof of Theorem 5 we can also extract the following multi-hop evaluation procedure. We do not specifically use this theorem for the applications explained in this work, but we nevertheless state it as it might be useful in the future.

**Corollary 6** (Multi-Hop Evaluation)**.** *Let $f = (f_0, \ldots, f_{r-1}) : \{0,1\}^d \to \mathbb{Z}_q^r$, where every $f_i$ is already represented in the normal form of Equation (24), and let $g : \{0,1\}^d \times \mathbb{Z}_q^r \to \mathbb{Z}_q$ be a depth-$\delta$ arithmetic circuit such that every intermediate wire in the computation of $g(\alpha, f(\alpha))$ has centered absolute value at most $B$, for every $\alpha$. There are polynomial-time algorithms*

- $\mathsf{TreeEvalM}\left(\mathsf{pk}, g, \{\mathsf{MBP}_{f_i}\}_{i\in[r]}\right) \mapsto \mathbf{M}_{g\circ f}$,
- $\mathsf{TreeEvalH}\left(\mathsf{pk}, g, \alpha, f(\alpha), \{\mathsf{MBP}_{f_i}\}_{i\in[r]}\right) \mapsto \left\{\mathbf{H}^{(i)}\right\}_{i\in[r+1]}$

*such that*

$$\mathbf{G}\mathbf{M}_{g\circ f}\overline{\mathbf{P}}_\alpha - g(\alpha, f(\alpha))\mathbf{G} = \sum_{i\in[r]} \left(\mathbf{G}\mathsf{MBP}_{f_i}(\alpha) - f_i(\alpha)\mathbf{G}\right)\mathbf{H}^{(i)} + \mathbf{Z}\mathbf{H}^{(r)}$$

*Furthermore,*

$$\begin{aligned}
\|\overline{\mathbf{P}}_\alpha^\intercal\|_{\infty,\infty} &\le n(mn)^d,\\
\|(\mathbf{H}^{(i)})^\intercal\|_{\infty,\infty} &\le n\left((tmn^4)^d + B\right)^\delta \qquad (i \in [r]),\\
\|(\mathbf{H}^{(r)})^\intercal\|_{\infty,\infty} &\le nt^2\left((tmn^4)^d + B\right)^\delta \cdot n^2\left(\left(n(tmn^4)^2\right)^d + \sum_{i\in[r]} \max_\beta \|\mathsf{MBP}_{f_i}(\beta)\|_{\infty,\infty}\right) \cdot \sum_{j=0}^{d-1} (t\|\mathbf{D}\|_{\infty,\infty})^j\,.
\end{aligned}$$

*Proof.* Use the branching programs $\mathsf{MBP}_{f_i}(\alpha)$ as the input wires and repeat the gate-by-gate construction from the proof of Theorem 5. In this construction, keep the corrections $\mathbf{H}_{f_i,\alpha}$ formal rather than substituting their values.

The update rules in Lemmas 8 to 10, together with flattening, imply inductively that the correction attached to every wire is an affine right-linear combination of the input corrections $\mathbf{H}_{f_i,\alpha}$. Its coefficient matrices and its constant term are computable from $(\mathsf{pk}, g, \alpha, f(\alpha), \{\mathsf{MBP}_{f_i}\}_{i\in[r]})$ alone. Indeed, the multiplication rule only additionally requires the values of intermediate wires, which are obtained by evaluating $g$ on the supplied value $(\alpha, f(\alpha))$, while each flattening correction contributes only to the constant term.

At the output wire, denote these coefficient matrices by $\{\mathbf{H}^{(i)}\}_{i\in[r]}$ and the constant term by $\mathbf{H}^{(r)}$. The final normalized branching program has the form $\mathbf{M}_{g\circ f}\overline{\mathbf{P}}_\alpha$, and the maintained key

equation gives

$$\begin{aligned}\mathbf{G}\left(\mathbf{M}_{g\circ f}\overline{\mathbf{P}}_\alpha - g(\alpha, f(\alpha))\mathbf{I}_n\right) &= \mathbf{Z}\left(\mathbf{H}^{(r)} + \sum_{i\in[r]} \mathbf{H}_{f_i,\alpha}\mathbf{H}^{(i)}\right)\\ &= \mathbf{Z}\mathbf{H}^{(r)} + \sum_{i\in[r]} \mathbf{G}\left(\mathsf{MBP}_{f_i}(\alpha) - f_i(\alpha)\mathbf{I}_n\right)\mathbf{H}^{(i)}.\end{aligned}$$

The norm analysis is the same as in the proof of Theorem 5. The flattening of the $i$-th input contributes at most

$$n^2 \max_\beta \|\mathsf{MBP}_{f_i}(\beta)\|_{\infty,\infty} \sum_{j=0}^{d-1} (t\|\mathbf{D}\|_{\infty,\infty})^j\,,$$

to the norm of $\mathbf{H}^{(r)}$. Each other flattening contributes at most

$$n^2\left(n(tmn^4)^2\right)^d \sum_{j=0}^{d-1} (t\|\mathbf{D}\|_{\infty,\infty})^j\,,$$

to the norm of $\mathbf{H}^{(r)}$. Each circuit layer enlarges the inherited correction by at most $(tmn^4)^d + B$. Unrolling the same recurrence bounds the row-sum norms of $\mathbf{H}^{(i)}$ and $\mathbf{H}^{(r)}$ by the stated right-hand sides divided by $n$ and $nt^2$, respectively. Since these matrices have $n$ and $nt^2$ rows, respectively, their maximum column sums are at most those factors times their row-sum norms, proving the transpose bounds. The bound for $\overline{\mathbf{P}}_\alpha^\intercal$ follows from the proof of Theorem 5. Polynomial-time computability follows from that of the construction in Theorem 5. □

Finally, recall that, except with negligible probability over the choice of $\mathsf{pk}$, $\|\mathbf{D}\|_{\infty,\infty} \le t\sigma\sqrt{\lambda}$, so we can simplify the expressions in the above bounds accordingly.

# 4 Laconic Shift-Hiding Pseudorandom Function

In the following we present a shift-hiding pseudorandom function (PRF) [PS18] that satisfies the *laconic property*, i.e., its master secret key does not depend on the complexity of the shift function (except for its circuit depth).

## 4.1 Laconic Function Evaluation

We recall the notion of laconic function evaluation LFE [QWW18]. For notational convenience, we are going to adopt a slightly modified syntax and security notion, which follow from routine modifications of existing work. To keep things simple, we are going to explicitly use the scheme from [QWW18] along with the GSW [GSW13] fully homomorphic encryption (FHE), although more recent constructions [AMR25b, AMR25a] would also work. We leave the exploration of different tradeoffs offered by different LFE scheme as ground for future work.

For powers of two $p \mid q$, we use $\mathbf{G}$ with the dimensions dictated by context: $k+1$ rows for GSW and homomorphic evaluation, and $k$ rows for the LFE hash and correctness equation. Write $\bar{\mathbf{G}}$ and $\mathbf{G}_{\text{last}}$ for the first $k$ rows and the last row of the GSW gadget; the GSW secret is $(\mathbf{s}^\intercal, -1)$.

Let $f = (f_0, \ldots, f_{\mu-1}) : \{0,1\}^\eta \to \mathbb{Z}_p^\mu$, with each $f_j(x)$ represented in $\{0, \ldots, p-1\}$, and let $\boldsymbol{\zeta}_i$ be the $i$-th standard basis vector of dimension $(k+1)\log q$. For $j \in [\mu]$, define

$$\mathbf{W}_f(\mathbf{c})_j = \sum_{a \in [\log p]} \mathsf{GSW.Eval}\left( x \mapsto \left\lfloor \frac{f_j(x)}{2^a} \right\rfloor \bmod 2, \mathbf{c} \right) \boldsymbol{\zeta}_{(k+1)\log q - \log p + a}$$

as the $j$-th column of $\mathbf{W}_f(\mathbf{c})$. Here $\mathsf{GSW.Eval}$ evaluates the indicated Boolean circuit on the input ciphertexts represented by $\mathbf{c}$. Thus $\mathbf{W}_f(\mathbf{c}) \in \mathbb{Z}_q^{(k+1)\times\mu}$ satisfies

$$(\mathbf{s}^\intercal, -1)\mathbf{W}_f(\mathbf{c}) = -\frac{q}{p} f(x)^\intercal + \mathbf{e}_{\mathrm{GSW}}^\intercal.$$

Let $\Psi_f$ be the Boolean circuit computing the bits of $\mathbf{W}_f$. Apply the standard homomorphic evaluation algorithms with gadget $\mathbf{G}$ to these output-bit circuits, using public matrices $[\hat{\mathbf{A}}_i^\intercal \mid \mathbf{0}]^\intercal$ and input vectors $\mathbf{b}_i + \mathbf{c}_i \mathbf{G}_{\mathrm{last}}$. For output bit $b$ of entry $(r, j)$, multiply its evaluated matrix and row vector by $\boldsymbol{\zeta}_{r\log q + b}$ and sum over $r \in [k+1]$, $b \in [\log q]$. Denote the resulting row by $\mathbf{b}_f^+ \in \mathbb{Z}_q^{1\times\mu}$ and write the resulting matrix as

$$\mathbf{A}_f^+ = \begin{bmatrix} \bar{\mathbf{A}}_f \\ \mathbf{a}_f^\intercal \end{bmatrix}, \qquad \bar{\mathbf{A}}_f \in \mathbb{Z}_q^{k\times\mu}, \quad \mathbf{a}_f \in \mathbb{Z}_q^\mu.$$

The standard evaluation equation gives

$$\mathbf{b}_f^+ = (\mathbf{s}^\intercal, -1)(\mathbf{A}_f^+ - \mathbf{W}_f(\mathbf{c})) + \mathbf{e}_{\mathrm{ev}}^\intercal.$$

The matrix $\mathbf{A}_f^+$ depends only on $\mathsf{crs}$ and $f$.

- $\mathsf{LFE.Setup}(1^\lambda, 1^\eta)$: The setup algorithm samples uniform matrices

  $$\hat{\mathbf{A}}_0, \ldots, \hat{\mathbf{A}}_{\tilde{\eta}-1} \in \mathbb{Z}_q^{k\times((k+1)\log q)}$$

  where $\tilde{\eta}$ denotes $\eta$ times the size of an FHE ciphertext, namely $\tilde{\eta} = \eta((k+1)\log q)^2$ bits. It outputs $\mathsf{crs} = (\hat{\mathbf{A}}_0, \ldots, \hat{\mathbf{A}}_{\tilde{\eta}-1})$.

- $\mathsf{LFE.Hash}(\mathsf{crs}, f)$: On input a function $f : \{0,1\}^\eta \to \mathbb{Z}_p^\mu$, the hashing algorithm computes

  $$\hat{\mathbf{A}}_f = \mathbf{G}^{-1}\left(\bar{\mathbf{A}}_f\right) \in \mathbb{Z}_q^{n\times\mu}$$

- $\mathsf{LFE.Encode}(\mathsf{crs}, \mathbf{s}, x)$: Use $(\mathbf{s}^\intercal, -1)$ as the GSW secret key. For all $i \in [\eta]$, let

  $$\mathbf{C}_i = \begin{bmatrix} \hat{\mathbf{B}}_i \\ \mathbf{s}^\intercal \hat{\mathbf{B}}_i + \mathbf{e}_i^\intercal \end{bmatrix} + x_i \mathbf{G}$$

  be a GSW encryption of $x_i$, for uniformly sampled $\hat{\mathbf{B}}_i \in \mathbb{Z}_q^{k\times((k+1)\log q)}$ and $\mathbf{e}_i \xleftarrow{\$} \mathcal{D}_{\bar{\sigma}}^{(k+1)\log q}$. Let $\mathbf{c} = \mathsf{vec}(\mathbf{G}^{-1}(\mathbf{C}_0, \ldots, \mathbf{C}_{\eta-1}))$. Then, for all $i \in [\tilde{\eta}]$, set

  $$\mathbf{b}_i = \mathbf{s}^\intercal(\hat{\mathbf{A}}_i - \mathbf{c}_i \cdot \bar{\mathbf{G}}) + \hat{\mathbf{e}}_i^\intercal$$

  where $\hat{\mathbf{e}}_i \xleftarrow{\$} \mathcal{D}_{\bar{\sigma}}^{(k+1)\log q}$. Output the encoding $\mathsf{ec} = (\mathbf{c}, \mathbf{b}_0, \ldots, \mathbf{b}_{\tilde{\eta}-1})$.

- $\mathsf{LFE.Eval}(\mathsf{crs}, \mathsf{ec}, f)$: Compute the evaluated row $\mathbf{b}_f^+$ as above and output $\mathbf{b}_f^+ + \mathbf{a}_f^\intercal$.

For correctness, we have that

$$\left\| \mathbf{s}^\intercal \mathbf{G} \hat{\mathbf{A}}_f + \frac{q}{p} f(x)^\intercal - \mathsf{LFE.Eval}(\mathsf{crs}, \mathsf{ec}, f) \right\|_\infty \leq \overline{\sigma} \cdot \lambda^{O(\delta)} \tag{27}$$

where $\delta \geq 1$ bounds the Boolean depth of $\Psi_f$; the bound holds except with negligible probability over the encoding noise. Indeed, adding $\mathbf{a}_f^\intercal$ cancels the last-row term in $(\mathbf{s}^\intercal, -1)\mathbf{A}_f^+$, and the two evaluation errors and the polynomially many output projections are bounded by $\overline{\sigma}\lambda^{O(\delta)}$. For security, we prove that the output of the encoding algorithm is pseudorandom, also in the presence of a decomposed LWE sample (see Section 2.4)

**Lemma 12** (Pseudorandomness)**.** *If the decomposed LWE assumption holds, then for all $x \in \{0,1\}^\eta$ the following distributions are computationally indistinguishable*

$$\{\mathsf{crs}, \mathsf{pk}, \mathbf{s}^\intercal \mathbf{Z} + \mathbf{e}^\intercal, \mathsf{ec}\} \approx_c \{\mathsf{crs}, \mathsf{pk}, \mathbf{u}_z, \mathbf{u}_{\mathsf{ec}}\}\,.$$

*Here* $\mathsf{pk} \xleftarrow{\$} \mathsf{TreeSetup}(1^\lambda)$, $\mathbf{Z}$ *is the matrix determined by* $\mathsf{pk}$, $\mathsf{crs} \xleftarrow{\$} \mathsf{LFE.Setup}(1^\lambda, 1^\eta)$, $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^k$, $\mathbf{e} \xleftarrow{\$} \mathcal{D}_{\overline{\sigma}}^{nt^2}$, *and* $\mathsf{ec} \xleftarrow{\$} \mathsf{LFE.Encode}(\mathsf{crs}, \mathbf{s}, x)$. *The variables* $\mathbf{u}_z$ *and* $\mathbf{u}_{\mathsf{ec}}$ *are independent uniform elements of the respective canonical representation spaces of* $\mathbf{s}^\intercal \mathbf{Z} + \mathbf{e}^\intercal$ *and* $\mathsf{ec}$.

*Proof.* For uniformly sampled $\hat{\mathbf{A}}_i^*$ we define $\hat{\mathbf{A}}_i = \hat{\mathbf{A}}_i^* + \mathbf{c}_i \cdot \bar{\mathbf{G}}$, where $\mathbf{c}$ is defined as in the encoding algorithm, which does not change the view of the distinguisher. Then the claim reduces to distinguishing

$$\mathbf{s}^\intercal \left[ \begin{array}{c|c|c|c|c|c} \mathbf{Z} & \hat{\mathbf{A}}_0^* & \ldots & \hat{\mathbf{A}}_{\tilde{\eta}-1}^* & \hat{\mathbf{B}}_0 & \ldots & \hat{\mathbf{B}}_{\eta-1} \end{array} \right] + \mathbf{E}$$

from uniform. This is precisely the decomposed LWE problem (Definition 1) with an auxiliary random matrix, which is as hard as the standard decomposed LWE [AMR25a]. □

### 4.2 Our Scheme

Let $f : \{0,1\}^d \times \{0,1\}^\eta \to \mathbb{Z}_p^\mu$ be a fixed shift function, where the polynomial output length $\mu = \mu(\lambda)$ is chosen by the application. We define the following additional parameters:

- The input length $\eta = \eta(\lambda)$.
- The rounding modulus $p = p(\lambda)$ divides $q$. We take both moduli to be powers of two and define $\lceil x \rfloor_p := \lfloor (p/q)x \rceil \bmod p$, coordinatewise. Thus translating by $(q/p)\mathbf{y}$ before rounding translates the output by $\mathbf{y} \in \mathbb{Z}_p^\mu$.
- Let $\delta$ and $B_g$ bound the depth and the centered wire values, respectively, of the arithmetic circuit computing

  $$g_{\mathsf{crs}}(\alpha) := \mathsf{vec}(\mathsf{LFE.Hash}(\mathsf{crs}, f_\alpha)).$$

  Choose $\delta \geq 1$ also to bound the depth of $\Psi_{f_\alpha}$ for every $\alpha$. Using the deterministic high-probability bound from Theorem 5, we choose the moduli so that

  $$\frac{q}{p} \geq 2^d \mu\, 2^{\omega(\log\lambda)} \cdot \left( \overline{\sigma}\sqrt{\lambda}\, n^2 t^2 \left( (tmn^4)^d + B_g \right)^\delta \cdot \left( 1 + n^2 \left( n(tmn^4)^2 \right)^d \right) \sum_{j=0}^{d-1} \left( t^2 \sigma \sqrt{\lambda} \right)^j + \overline{\sigma}\lambda^{O(\delta)} \right).$$

For a Boolean circuit of depth $\delta_f$ computing $f$, standard parallel matrix arithmetic gives a common bound $\delta = \delta_f(\log k + \log\log q)^{O(1)}$ for these depths.

We describe our scheme in the following.

- $\mathsf{Setup}(1^\lambda, 1^\eta)$: Sample

$$\mathsf{pk} \xleftarrow{\$} \mathsf{TreeSetup}(1^\lambda) \qquad \mathsf{crs} \xleftarrow{\$} \mathsf{LFE.Setup}(1^\lambda, 1^\eta) \qquad \mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^k \qquad \mathbf{e} \xleftarrow{\$} \mathcal{D}_{\overline{\sigma}}^{nt^2} \qquad \mathbf{t} \xleftarrow{\$} \mathbb{Z}_q^\mu$$

  and define $\mathbf{z}^\intercal = \mathbf{s}^\intercal\mathbf{Z} + \mathbf{e}^\intercal$. Let $f_\alpha$ denote the function $f$ with $\alpha$ hardwired as its first input. Compute

$$\tilde{\mathbf{M}} = \mathsf{TreeEvalM}(\mathsf{pk}, g_{\mathsf{crs}}).$$

  Output $(K, \mathsf{crs})$, where $K = (\mathsf{pk}, \mathbf{z}, \tilde{\mathbf{M}}, \mathbf{s}, \mathbf{t})$.

- $\mathsf{Shift}(\mathsf{crs}, K, \beta)$: Compute

$$\mathsf{ec} \xleftarrow{\$} \mathsf{LFE.Encode}(\mathsf{crs}, \mathbf{s}, \beta)$$

  and output the shifted key $K_\beta = (\mathsf{pk}, \mathsf{crs}, \mathbf{z}, \mathsf{ec}, \mathbf{t})$.

- $\mathsf{Eval}(K, \alpha)$: Output

$$\left\lceil \mathbf{s}^\intercal\mathbf{G}\tilde{\mathbf{M}} \cdot \overline{\mathbf{P}}_\alpha \cdot \mathbf{V}' + \mathbf{t}^\intercal \right\rfloor_p$$

  where $\overline{\mathbf{P}}_\alpha = \mathsf{TreeExpand}(\mathsf{pk}, 1^{n\mu}, \alpha)$ is the $n\mu$-output expansion and $\mathbf{V}' \in \{0,1\}^{n^2\mu \times \mu}$ is the fixed matrix satisfying

$$(\mathsf{vec}(\mathbf{A})^\intercal \otimes \mathbf{I}_n)\mathbf{V}' = \mathbf{A} \qquad \text{for every } \mathbf{A} \in \mathbb{Z}_q^{n\times\mu}.$$

  In particular, $\|(\mathbf{V}')^\intercal\|_{\infty,\infty} = n$.

- $\mathsf{ShiftEval}(K_\beta, \alpha)$: Compute

$$\mathbf{c}^\intercal = \mathsf{LFE.Eval}(\mathsf{crs}, \mathsf{ec}, f_\alpha).$$

  Derive $\mathbf{H}_{g_{\mathsf{crs}},\alpha} = \mathsf{TreeEvalH}(\mathsf{pk}, g_{\mathsf{crs}}, \alpha)$ and output

$$\left\lceil \mathbf{z}^\intercal \cdot \mathbf{H}_{g_{\mathsf{crs}},\alpha} \cdot \mathbf{V}' + \mathbf{c}^\intercal + \mathbf{t}^\intercal \right\rfloor_p$$

  where the rounding modulo $p$ operation is extended to vectors component-wise.

Observe that $\mathsf{Eval}$ can be computed by a polynomial-size Boolean circuit of depth $\mathsf{poly}(\log(\delta), \log\lambda)$, whereas $\mathsf{ShiftEval}$ can be computed by a polynomial-size Boolean circuit of depth $(\delta) \cdot \mathsf{poly}(\log\lambda)$. We show that the scheme satisfies correctness, that is, the output of the shifted key equals the output of the evaluation algorithm, shifted by the value dictated by $f$.

**Lemma 13** (Correctness)**.** *There exists a negligible function $\nu$ such that for all $\lambda \in \mathbb{N}$ and all $\beta \in \{0,1\}^\eta$ we have*

$$\Pr\left(\exists\alpha : \mathsf{ShiftEval}(K_\beta, \alpha) \neq \mathsf{Eval}(K, \alpha) + f(\alpha, \beta)\right) \leq \nu(\lambda).$$

*Proof.* By Equation (27) we have that

$$\mathbf{c}^\intercal = \mathbf{s}^\intercal \mathbf{G} \hat{\mathbf{A}}_{f_\alpha} + \frac{q}{p} f(\alpha, \beta)^\intercal + \hat{\mathbf{e}}^\intercal \tag{28}$$

where $\hat{\mathbf{A}}_{f_\alpha} = \mathsf{LFE.Hash}(\mathsf{crs}, f_\alpha)$ and $\|\hat{\mathbf{e}}\|_\infty \leq \overline{\sigma} \cdot \lambda^{O(\delta)}$. By Theorem 5, we have that

$$\mathbf{Z} \cdot \mathbf{H}_{g_{\mathsf{crs}},\alpha} = \mathbf{G}(\tilde{\mathbf{M}} \cdot \overline{\mathbf{P}}_\alpha - g_{\mathsf{crs}}(\alpha)^\intercal \otimes \mathbf{I}_n), \qquad g_{\mathsf{crs}}(\alpha) = \mathsf{vec}(\hat{\mathbf{A}}_{f_\alpha}),$$

where, except with negligible probability over $\mathsf{pk}$ and $\mathbf{e}$,

$$\|\mathbf{H}^\intercal_{g_{\mathsf{crs}},\alpha}\|_{\infty,\infty} \leq nt^2 \left((tmn^4)^d + B_g\right)^\delta \left(1 + n^2 \left(n(tmn^4)^2\right)^d\right) \sum_{j=0}^{d-1} \left(t^2 \sigma \sqrt{\lambda}\right)^j.$$

We also have $\|\mathbf{e}\|_\infty \leq \overline{\sigma}\sqrt{\lambda}$ on the same event. Therefore,

$$\mathbf{Z} \cdot \mathbf{H}_{g_{\mathsf{crs}},\alpha} \cdot \mathbf{V}' = \mathbf{G}\tilde{\mathbf{M}} \cdot \overline{\mathbf{P}}_\alpha \cdot \mathbf{V}' - \mathbf{G}\hat{\mathbf{A}}_{f_\alpha}.$$

We obtain that

$$\begin{aligned}
\mathbf{z}^\intercal \cdot \mathbf{H}_{g_{\mathsf{crs}},\alpha} \cdot \mathbf{V}' + \mathbf{c}^\intercal + \mathbf{t}^\intercal &= \mathbf{s}^\intercal \mathbf{G}\tilde{\mathbf{M}} \cdot \overline{\mathbf{P}}_\alpha \cdot \mathbf{V}' - \mathbf{s}^\intercal \mathbf{G}\hat{\mathbf{A}}_{f_\alpha} + \mathbf{e}^\intercal \cdot \mathbf{H}_{g_{\mathsf{crs}},\alpha} \cdot \mathbf{V}' + \mathbf{c}^\intercal + \mathbf{t}^\intercal \\
&= \mathbf{s}^\intercal \mathbf{G}\tilde{\mathbf{M}} \cdot \overline{\mathbf{P}}_\alpha \cdot \mathbf{V}' + \frac{q}{p} f(\alpha, \beta)^\intercal + \mathbf{e}^\intercal \cdot \mathbf{H}_{g_{\mathsf{crs}},\alpha} \cdot \mathbf{V}' + \hat{\mathbf{e}}^\intercal + \mathbf{t}^\intercal
\end{aligned}$$

We can then bound

$$\begin{aligned}
&\|\mathbf{e}^\intercal \mathbf{H}_{g_{\mathsf{crs}},\alpha} \cdot \mathbf{V}' + \hat{\mathbf{e}}^\intercal\|_\infty \\
&\leq \|\mathbf{e}^\intercal \mathbf{H}_{g_{\mathsf{crs}},\alpha} \cdot \mathbf{V}'\|_\infty + \|\hat{\mathbf{e}}\|_\infty \\
&\leq \|\mathbf{e}\|_\infty \|\mathbf{H}^\intercal_{g_{\mathsf{crs}},\alpha}\|_{\infty,\infty} \|(\mathbf{V}')^\intercal\|_{\infty,\infty} + \|\hat{\mathbf{e}}\|_\infty \\
&\leq \overline{\sigma}\sqrt{\lambda}\, n^2 t^2 \left((tmn^4)^d + B_g\right)^\delta \cdot \left(1 + n^2 \left(n(tmn^4)^2\right)^d\right) \sum_{j=0}^{d-1} \left(t^2 \sigma \sqrt{\lambda}\right)^j + \overline{\sigma}\lambda^{O(\delta)}.
\end{aligned}$$

Using the uniform mask $\mathbf{t}$ and the modulus condition above, for every fixed input-coordinate pair $(\alpha, j)$ the probability that the noise crosses a rounding boundary is at most $\nu(\lambda) \cdot 2^{-d}/\mu$, for a negligible $\nu$. A union bound over all $2^d\mu$ pairs proves the claim. □

We note that the construction is *laconic*, in the sense that the complexity of the $\mathsf{Shift}$ and $\mathsf{Eval}$ algorithms depends (polynomially) on the depth of the circuit computing $f$ and on the size of its input, but it is otherwise independent of the circuit complexity of $f$. Furthermore the circuit depth of the $\mathsf{Eval}$ algorithm can be bounded by a fixed polynomial in the security parameter.

Finally, we show that our construction satisfies shift hiding. In fact we prove a slightly stronger property, which immediately implies the desired statement.

**Lemma 14** (Shift Hiding)**.** *If the decomposed LWE assumption holds, then for every $\beta \in \{0,1\}^\eta$,*

$$\left\{\mathsf{Shift}(\mathsf{crs}, K, \beta) : (K, \mathsf{crs}) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, 1^\eta)\right\} \approx_c \mathcal{U}_{\mathsf{shift}},$$

*where $\mathcal{U}_{\mathsf{shift}}$ is sampled by drawing $\mathsf{pk} \xleftarrow{\$} \mathsf{TreeSetup}(1^\lambda)$ and $\mathsf{crs} \xleftarrow{\$} \mathsf{LFE.Setup}(1^\lambda, 1^\eta)$ as in the real scheme, and then sampling the remaining components $(\mathbf{u}_z, \mathbf{u}_{\mathsf{ec}}, \mathbf{t})$ independently and uniformly from the canonical representation spaces of $(\mathbf{z}, \mathsf{ec}, \mathbf{t})$.*

*Proof.* Follows as an immediate consequence of Lemma 12 and by definition of each component of the shifted key. □

### 4.3 Removing the Input Dependence

In the laconic shift-hiding function we have just described, the size of the keys (both shifted and unshifted) grows polynomially in the size of $\alpha$ due to the modulus dependency. Next, we show how to remove this dependence using trapdoor hashing. We recall the syntax and properties of this primitive.

**Definition 2** (Trapdoor Hashing)**.** *Let $f : \{0,1\}^d \times \{0,1\}^\eta \to \mathbb{Z}_p^\mu$ be a function. A trapdoor hash function for $f$ consists of a tuple $\mathsf{TDH} = (\mathsf{Setup}, \mathsf{Hash}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ with the following syntax:*

- $\mathsf{TDH.Setup}(1^\lambda)$*: The setup is randomised, takes as input the security parameter $1^\lambda$, and outputs a hash key $\mathsf{hk}$.*
- $\mathsf{TDH.Hash}(\mathsf{hk}, \alpha)$*: The hash is deterministic and takes as input the hash key $\mathsf{hk}$ and a string $\alpha \in \{0,1\}^d$. The output is a digest $h \in \{0,1\}^k$ where $k = \mathsf{poly}(\lambda, \log d, \log \eta)$.*
- $\mathsf{TDH.Gen}(\mathsf{hk}, \beta)$*: The generation algorithm takes as input the hash key $\mathsf{hk}$ and a string $\beta \in \{0,1\}^\eta$. The output is an encoding key $\mathsf{ek}$ and a trapdoor $\mathsf{td} \in \{0,1\}^\eta$.*
- $\mathsf{TDH.Enc}(\mathsf{hk}, \mathsf{ek}, \alpha)$*: The encoding algorithm is deterministic and takes as input the hash key $\mathsf{hk}$, the encoding key $\mathsf{ek}$ and a string $\alpha \in \{0,1\}^d$. The output is an encoding $\mathbf{e} \in \mathbb{Z}_p^\mu$.*
- $\mathsf{TDH.Dec}(\mathsf{hk}, \mathsf{td}, h)$*: The decoding algorithm is deterministic and takes as input the hash key $\mathsf{hk}$, the trapdoor $\mathsf{td}$ and a digest $h$. The output is an encoding $\mathbf{e}' \in \mathbb{Z}_p^\mu$.*

The primitive satisfies the following correctness property.

**Definition 3** (Correctness of Trapdoor Hashing)**.** *A trapdoor hash function $\mathsf{TDH}$ for $f$ is correct if there exists a negligible function $\mathsf{negl}(\lambda)$ such that, for every $\lambda \in \mathbb{N}$ and input $\beta \in \{0,1\}^\eta$, it holds that*

$$\Pr\left[\exists \alpha : \mathsf{Enc}(\mathsf{hk}, \mathsf{ek}, \alpha) + \mathsf{Dec}(\mathsf{hk}, \mathsf{td}, h) \neq f(\alpha, \beta)\right] \leq \mathsf{negl}(\lambda),$$

*where the probability is taken over the randomness of $\mathsf{hk} \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$ and $(\mathsf{ek}, \mathsf{td}) \xleftarrow{\$} \mathsf{Gen}(\mathsf{hk}, \beta)$ and $h = \mathsf{Hash}(\mathsf{hk}, \alpha)$.*

Moreover, we require the following security property.

**Definition 4** (Security of Trapdoor Hashing)**.** *A trapdoor hash function $\mathsf{TDH}$ for $f$ is secure if there exists a negligible function $\mathsf{negl}(\lambda)$ such that, for every PPT adversary $\mathcal{A}$, every sufficiently large $\lambda \in \mathbb{N}$ and every $\beta_0, \beta_1 \in \{0,1\}^\eta$, it holds that*

$$\left|\Pr\left[\mathcal{A}(1^\lambda, \mathsf{hk}, \mathsf{ek}_1) = 1\right] - \Pr\left[\mathcal{A}(1^\lambda, \mathsf{hk}, \mathsf{ek}_0) = 1\right]\right| \leq \mathsf{negl}(\lambda)$$

*where the probabilities are taken over the randomness of $\mathsf{hk} \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$, $(\mathsf{ek}_0, \mathsf{td}_0) \xleftarrow{\$} \mathsf{Gen}(\mathsf{hk}, \beta_0)$, $(\mathsf{ek}_1, \mathsf{td}_1) \xleftarrow{\$} \mathsf{Gen}(\mathsf{hk}, \beta_1)$ and of the adversary $\mathcal{A}$.*

We then recall the following theorem.

**Theorem 7** ([AJSW26])**.** *Under the hardness of LWE, there exists a trapdoor hashing scheme where the encoding key has size $\eta + D \cdot \mathsf{poly}(\lambda)$ where $D$ is the depth of the Boolean circuit describing $f$ and the hashing algorithm has circuit size $\mathsf{poly}(d, \lambda)$ and depth $\mathsf{poly}(\log d, \log \lambda)$. Moreover, the encoding key is computationally indistinguishable from a uniformly random string.*

Let $\mathsf{TDH}$ be a trapdoor hashing scheme for the function $f$ we want to address in our input-succinct laconic shift hiding construction, and let $\mathsf{SH}' = (\mathsf{Setup}, \mathsf{Shift}, \mathsf{Eval}, \mathsf{ShiftEval})$ be another (non input succinct) laconic shift-hiding construction for the function

$$f'_{\mathsf{hk}} : \{0,1\}^k \times \{0,1\}^\eta \longrightarrow \mathbb{Z}_p^\mu$$
$$(h, \mathsf{td}) \longmapsto \mathsf{TDH.Dec}(\mathsf{hk}, \mathsf{td}, h)$$

Our new shift-hiding function will be as follows:

- $\mathsf{Setup}(1^\lambda, 1^\eta)$: Sample $\mathsf{hk} \xleftarrow{\$} \mathsf{TDH.Setup}(1^\lambda)$, then $(K', \mathsf{crs}) \xleftarrow{\$} \mathsf{SH}'.\mathsf{Setup}(1^\lambda, 1^\eta)$ for $f'_{\mathsf{hk}}$ and $\mathbf{t} \xleftarrow{\$} \mathbb{Z}_p^\mu$. Output $(K, \mathsf{crs})$, where $K = (K', \mathsf{hk}, \mathbf{t})$.
- $\mathsf{Shift}(\mathsf{crs}, K, \beta)$: Compute $(\mathsf{ek}, \mathsf{td}) \xleftarrow{\$} \mathsf{TDH.Gen}(\mathsf{hk}, \beta)$ and $K'_{\mathsf{td}} \xleftarrow{\$} \mathsf{SH}'.\mathsf{Shift}(\mathsf{crs}, K', \mathsf{td})$. Output $K_\beta = (\mathsf{hk}, \mathsf{ek}, K'_{\mathsf{td}}, \mathbf{t})$.
- $\mathsf{Eval}(K, \alpha)$: Compute $h = \mathsf{TDH.Hash}(\mathsf{hk}, \alpha)$ and output $\mathsf{SH}'.\mathsf{Eval}(K', h) + \mathbf{t}$.
- $\mathsf{ShiftEval}(K_\beta, \alpha)$: Compute $h = \mathsf{TDH.Hash}(\mathsf{hk}, \alpha)$, then output
$$\mathsf{SH}'.\mathsf{ShiftEval}(K'_{\mathsf{td}}, h) + \mathsf{TDH.Enc}(\mathsf{hk}, \mathsf{ek}, \alpha) + \mathbf{t}.$$

It is easy to see that the new laconic shift-hiding function satisfies the required correctness properties. Indeed, by the correctness of $\mathsf{SH}'$, we have that for all $h$, except with $\mathsf{negl}(\lambda)$ probability,

$$\mathsf{SH}'.\mathsf{ShiftEval}(K'_{\mathsf{td}}, h) = \mathsf{SH}'.\mathsf{Eval}(K', h) + f'_{\mathsf{hk}}(h, \mathsf{td}) = \mathsf{Eval}(K, \alpha) - \mathbf{t} + \mathsf{TDH.Dec}(\mathsf{hk}, \mathsf{td}, h).$$

Moreover, by the correctness of $\mathsf{TDH}$, we have that for all $\alpha$, except with $\mathsf{negl}(\lambda)$ probability,

$$\mathsf{SH}'.\mathsf{ShiftEval}(K'_{\mathsf{td}}, h) + \mathsf{TDH.Enc}(\mathsf{hk}, \mathsf{ek}, \alpha) + \mathbf{t} = \mathsf{Eval}(K, \alpha) + f(\alpha, \beta).$$

The shift-hiding property is also easy to verify. The shift hiding property of $\mathsf{SH}'$ ensures that $K'_{\mathsf{td}}$ is indistinguishable from random. Then, given that $\mathsf{td}$ is kept secret, we can rely on security of trapdoor hashing to argue that $\mathsf{ek}$ looks random too.

If we instantiate $\mathsf{TDH}$ using the construction of [AJSW26] and $\mathsf{SH}'$ with the construction in the previous section, we obtain a laconic shift-hiding function where the keys are independent of the size of the first input $\alpha$.

**Intractable Laconic Shift-Hiding PRF.** Some of our results will later need a laconic shift-hiding PRF where it is hard for the adversary to find an input $\alpha$ such that, for any entry $i$, we have

$$\left| \left( \mathsf{Eval}(K, \alpha)_i - \frac{p}{2} \right) \bmod p \right| < B$$

where $\mathsf{Eval}(K, \alpha)_i$ denotes the $i$-th entry of $\mathsf{Eval}(K, \alpha)$. If a construction satisfies this property, we call it $B$-intractable. It is easy to prove that the input-succinct laconic shift-hiding function we presented is $B$-intractable provided that $p > 2^{k+\omega(\log \lambda)} B$. As a matter of fact, the image of $\mathsf{Eval}(K, \cdot)$ contains at most $2^k$ elements: one for each possible $\mathsf{TDH}$ digest. Notice that the probability over $K$ that, for a fixed $\alpha$,

$$\left| \left( \mathsf{Eval}(K, \alpha)_i - \frac{p}{2} \right) \bmod p \right| < B$$

is exactly $2B/p$ due to the random pad $\mathbf{t}$. Union bounding over possible $\mathsf{TDH}$ hashes and indexes $i$, we obtain that the probability that any bad pair $(\alpha, i)$ exists is at most $\mathsf{poly}(\lambda) \cdot 2^k B/p$. Notice that when $p$ is chosen as we proposed, this probability is negligible.

### 4.4 A Heuristic Approach to Obfuscation

We discuss a simple approach to construct an obfuscator for all circuits, as a simple application of our construction in Section 4.2. To obfuscate a circuit $C$, we define the constraint $\beta_K$ and the function $f$ to be such that

$$f(\alpha, \beta_K) = C(\alpha) - \mathsf{Eval}(K, \alpha)$$

and we let the obfuscation be the key shifted at $\beta_K$. One can evaluate the obfuscated circuit by running

$$\mathsf{ShiftEval}(K_{\beta_K}, \alpha) = \mathsf{Eval}(K, \alpha) + C(\alpha) - \mathsf{Eval}(K, \alpha) = C(\alpha)$$

by Lemma 13. We note that this construction is only well-defined because of the laconic property of the scheme, since the complexity of the $\mathsf{Eval}$ algorithm only depends on the depth of the constrained function and moreover the depth of $\mathsf{Eval}$ is bounded by a polynomial in $\lambda$ (regardless of the constraint).

Unfortunately, the shift hiding property of our laconic shift-hiding PRF (Lemma 14) holds for any fixed constraint $\beta$, but the proof strategy fails if $\beta$ is allowed to depend on $K$. Nevertheless, we view this circularity as a reasonable heuristic for security, and we conjecture that the obfuscator described above is secure. We leave the analysis of such construction as a fascinating open problem.

## 5 Compressing LWE Samples

In the following we show how to give *compressed* LWE samples with respect to a particular class of matrices, which later on can be expanded to fully-fledged LWE samples. Before presenting our construction, we define the *block-transpose* operator $\Pi_k$. Let $\mathbf{M} \in \mathbb{Z}_q^{k \cdot n_0 \times n_1}$ be a matrix, and write $\mathbf{M}$ as a block matrix of $k$-dimensional vectors $\mathbf{m}_{i,j} \in \mathbb{Z}_q^k$:

$$\mathbf{M} = \{\mathbf{m}_{i,j}\}_{i,j} = \begin{bmatrix} \mathbf{m}_{0,0} & \cdots & \mathbf{m}_{0,n_1-1} \\ \vdots & \ddots & \vdots \\ \mathbf{m}_{n_0-1,0} & \cdots & \mathbf{m}_{n_0-1,n_1-1} \end{bmatrix}.$$

We define the action of the operator $\Pi_k$ as

$$\Pi_k(\mathbf{M}) = \{\mathbf{m}_{j,i}\}_{i,j} = \begin{bmatrix} \mathbf{m}_{0,0} & \cdots & \mathbf{m}_{n_0-1,0} \\ \vdots & \ddots & \vdots \\ \mathbf{m}_{0,n_1-1} & \cdots & \mathbf{m}_{n_0-1,n_1-1} \end{bmatrix} \in \mathbb{Z}_q^{k \cdot n_1 \times n_0}.$$

In other words, the output of the operator is a matrix where the blocks are transposed, where each block is a $k$-dimensional vector. It can be verified by direct calculation that $\Pi_k$ is linear and that, for all vectors $\mathbf{s} \in \mathbb{Z}_q^k$ and matrices $\mathbf{M} \in \mathbb{Z}_q^{k \cdot n_0 \times n_1}$, it holds that

$$\Pi_k(\Pi_k(\mathbf{M})) = \mathbf{M} \qquad \text{and} \qquad (\mathbf{I}_{n_1} \otimes \mathbf{s}^\intercal) \cdot \Pi_k(\mathbf{M}) = ((\mathbf{I}_{n_0} \otimes \mathbf{s}^\intercal) \cdot \mathbf{M})^\intercal. \tag{29}$$

We also introduce the following two lemmas, which allow us to construct matrix trapdoors by appropriately combining matrices.

**Lemma 15** (Matrix Trapdoors)**.** *Consider any matrices* $\mathbf{A} \in \mathbb{Z}_q^{t\cdot k\times m}$ *and* $\mathbf{B} \in \mathbb{Z}_q^{m\times t/n}$*. Let*

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_0 \\ \vdots \\ \mathbf{A}_{t-1} \end{bmatrix}$$

*be the blocks of* $\mathbf{A}$ *and define*

$$\mathbf{U} = \Pi_k(\mathbf{AB}) + \mathbf{I}_{t/n} \otimes \mathbf{G}, \qquad \mathbf{L} = -\mathbf{G}^{-1}\left[\ \mathsf{vec}(\mathbf{A}_0) \ \middle|\ \cdots \ \middle|\ \mathsf{vec}(\mathbf{A}_{t-1})\ \right].$$

*Then, it holds that* $\|\mathbf{L}\|_\infty \leq 1$ *and*

$$\left[\ \mathbf{U} \ \middle|\ \mathbf{B}^\intercal \otimes \mathbf{G}\ \right] \cdot \begin{bmatrix} \mathbf{I}_t \\ \mathbf{L} \end{bmatrix} = \left[\ \mathbf{U} \ \middle|\ \mathbf{B}^\intercal \otimes \mathbf{I}_k\ \right] \cdot \begin{bmatrix} \mathbf{I}_t \\ (\mathbf{I}_m \otimes \mathbf{G})\mathbf{L} \end{bmatrix} = \mathbf{I}_{t/n} \otimes \mathbf{G}, \tag{30}$$

*Proof.* The first identity in Equation (30) follows by Equation (9) since

$$(\mathbf{B}^\intercal \otimes \mathbf{G})\mathbf{L} = (\mathbf{B}^\intercal \otimes \mathbf{I}_k) \cdot (\mathbf{I}_m \otimes \mathbf{G})\mathbf{L}.$$

For the second identity, observe that

$$(\mathbf{I}_m \otimes \mathbf{G})\mathbf{L} = -\left[\ \mathsf{vec}(\mathbf{A}_0) \ \middle|\ \cdots \ \middle|\ \mathsf{vec}(\mathbf{A}_{t-1})\ \right].$$

Moreover, by Equation (10)

$$(\mathbf{B}^\intercal \otimes \mathbf{I}_k) \cdot \mathsf{vec}(\mathbf{A}_i) = \mathsf{vec}(\mathbf{A}_i\mathbf{B}).$$

Finally, by definition of the block-transpose operator

$$\Pi_k(\mathbf{AB}) = \Pi_k \begin{bmatrix} \mathbf{A}_0\mathbf{B} \\ \vdots \\ \mathbf{A}_{t-1}\mathbf{B} \end{bmatrix} = \left[\ \mathsf{vec}(\mathbf{A}_0\mathbf{B}) \ \middle|\ \cdots \ \middle|\ \mathsf{vec}(\mathbf{A}_{t-1}\mathbf{B})\ \right].$$

Putting all these observations together, we derive Equation (30). □

The following is a recursive version of the lemma above.

**Lemma 16** (Recursive Matrix Trapdoor)**.** *Consider any matrices* $\mathbf{A} \in \mathbb{Z}_q^{t\cdot k\times m}$ *and* $\mathbf{B} \in \mathbb{Z}_q^{m\times t/n}$ *and define* $\mathbf{U}$ *and* $\mathbf{L}$ *as in Lemma 15. Set* $c = t/(nm)$ *and*

$$\mathbf{U}_0 = \mathbf{U}, \qquad \mathbf{B}_0 = \mathbf{B}, \qquad \mathbf{L}_0 = \mathbf{L}.$$

*Then, inductively define* $\mathbf{U}_i \in \mathbb{Z}_q^{m\cdot c^{i+1}\cdot k\times t\cdot c^i}, \mathbf{B}_i \in \mathbb{Z}_q^{m\times m\cdot c^{i+1}}, \mathbf{L}_i \in \mathbb{Z}_q^{n\cdot m\times n\cdot m\cdot c^{i+1}}$ *as*

$$\begin{aligned} \mathbf{U}_{i+1} &= \mathbf{I}_c \otimes \mathbf{U}_i + (\mathbf{I}_c \otimes \mathbf{B}_i^\intercal \otimes \mathbf{I}_k) \cdot \mathbf{U} \cdot (\mathbf{I}_c \otimes \mathbf{L}_i) \\ \mathbf{B}_{i+1} &= \mathbf{B} \cdot (\mathbf{I}_c \otimes \mathbf{B}_i) \\ \mathbf{L}_{i+1} &= \mathbf{L} \cdot (\mathbf{I}_c \otimes \mathbf{L}_i). \end{aligned}$$

*For every* $i \in \mathbb{N}$*, it holds that* $\|\mathbf{L}_i\|_\infty \leq (nm)^i$*,* $\|\mathbf{B}_i\|_\infty \leq \|\mathbf{B}\|_\infty(m\|\mathbf{B}\|_\infty)^i$*, and*

$$\left[\ \mathbf{U}_i \ \middle|\ \mathbf{B}_i^\intercal \otimes \mathbf{G}\ \right] \cdot \begin{bmatrix} \mathbf{I}_{t\cdot c^i} \\ \mathbf{L}_i \end{bmatrix} = \left[\ \mathbf{U}_i \ \middle|\ \mathbf{B}_i^\intercal \otimes \mathbf{I}_k\ \right] \cdot \begin{bmatrix} \mathbf{I}_{t\cdot c^i} \\ (\mathbf{I}_m \otimes \mathbf{G})\mathbf{L}_i \end{bmatrix} = \mathbf{I}_{m\cdot c^{i+1}} \otimes \mathbf{G} \tag{31}$$

*Proof.* The norm bounds follow by a straightforward induction and the first identity in Equation (31) holds by the same reasoning as in Lemma 15. To prove the second identity of Equation (31) we proceed with an induction argument over $i$. The base case ($i = 0$) is precisely proven in Lemma 15. As for the inductive step, observe that:

$$
\begin{aligned}
&\left[\ \mathbf{U}_{i+1} \ \middle|\ \mathbf{B}_{i+1}^{\intercal} \otimes \mathbf{G}\ \right] \cdot \begin{bmatrix} \mathbf{I}_{t \cdot c^{i+1}} \\ \mathbf{L}_{i+1} \end{bmatrix} \\
&= \mathbf{U}_{i+1} + (\mathbf{B}_{i+1}^{\intercal} \otimes \mathbf{G}) \cdot \mathbf{L}_{i+1} \\
&= \mathbf{U}_{i+1} + \Big(((\mathbf{I}_c \otimes \mathbf{B}_i^{\intercal}) \cdot \mathbf{B}^{\intercal}) \otimes \mathbf{G}\Big) \cdot \mathbf{L} \cdot (\mathbf{I}_c \otimes \mathbf{L}_i) \\
&= \mathbf{U}_{i+1} + (\mathbf{I}_c \otimes \mathbf{B}_i^{\intercal} \otimes \mathbf{I}_k) \cdot (\mathbf{B}^{\intercal} \otimes \mathbf{G}) \cdot \mathbf{L} \cdot (\mathbf{I}_c \otimes \mathbf{L}_i) \\
&= \mathbf{I}_c \otimes \mathbf{U}_i + (\mathbf{I}_c \otimes \mathbf{B}_i^{\intercal} \otimes \mathbf{I}_k) \cdot \mathbf{U} \cdot (\mathbf{I}_c \otimes \mathbf{L}_i) + (\mathbf{I}_c \otimes \mathbf{B}_i^{\intercal} \otimes \mathbf{I}_k) \cdot (\mathbf{B}^{\intercal} \otimes \mathbf{G}) \cdot \mathbf{L} \cdot (\mathbf{I}_c \otimes \mathbf{L}_i) \\
&= \mathbf{I}_c \otimes \mathbf{U}_i + (\mathbf{I}_c \otimes \mathbf{B}_i^{\intercal} \otimes \mathbf{I}_k) \cdot \Big(\mathbf{U} + (\mathbf{B}^{\intercal} \otimes \mathbf{G}) \cdot \mathbf{L}\Big) \cdot (\mathbf{I}_c \otimes \mathbf{L}_i) \\
&= \mathbf{I}_c \otimes \mathbf{U}_i + (\mathbf{I}_c \otimes \mathbf{B}_i^{\intercal} \otimes \mathbf{I}_k) \cdot (\mathbf{I}_{m \cdot c} \otimes \mathbf{G}) \cdot (\mathbf{I}_c \otimes \mathbf{L}_i) \\
&= \mathbf{I}_c \otimes \Big(\mathbf{U}_i + (\mathbf{B}_i^{\intercal} \otimes \mathbf{I}_k) \cdot (\mathbf{I}_m \otimes \mathbf{G}) \cdot \mathbf{L}_i\Big) \\
&= \mathbf{I}_c \otimes \mathbf{I}_{m \cdot c^{i+1}} \otimes \mathbf{G}
\end{aligned}
$$

where the last identity follows by induction hypothesis. This concludes our proof. □

### 5.1 The Compression Algorithm

We describe the scheme along with its correctness proof. Later on, we will show how to sample the inputs to the algorithms so that we can prove its security. We introduce new parameters:

- $s = s(\lambda)$ be the depth of the recursion.
- $\sigma' = \sigma'(\lambda)$ be another Gaussian parameter.
- $c > 1$ a constant.

We describe the algorithms of our scheme in the following.

- $\mathsf{Setup}(1^\lambda)$: Sample $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{tk \times m}$ and $\tilde{\mathbf{B}} \xleftarrow{\$} \mathcal{D}_\sigma^{m \times tn}$. Define $\mathbf{B} = \tilde{\mathbf{B}}\mathbf{G}^{-1}(\Pi_k(\mathbf{I}_{t/n} \otimes \mathbf{G}))$. Output $\mathsf{hk} = (\mathbf{A}, \mathbf{B})$.
- $\mathsf{Hash}(\mathsf{hk}, \mathbf{X}, \mathbf{w})$: Suppose that $\mathbf{X}$ has $k$ rows and $m \cdot c^s$ columns. Set $\mathbf{x} = \mathsf{vec}(\mathbf{X})$ and compute

$$
\begin{bmatrix} \mathbf{w}_0 \\ \mathbf{w}_1 \end{bmatrix} = \mathsf{ShiftedPreimage}\left(\left[\ \mathbf{U}_{s-1} \ \middle|\ \mathbf{B}_{s-1}^{\intercal} \otimes \mathbf{G}\ \right], \begin{bmatrix} \mathbf{I}_{t \cdot c^{s-1}} \\ \mathbf{L}_{s-1} \end{bmatrix}, \mathbf{x}, \mathbf{w}\right)
$$

  with variables defined as in Lemma 16. Output the hash $\mathbf{d} = (\mathbf{I}_m \otimes \mathbf{G}) \cdot \mathbf{w}_1$ and the expander key $\mathsf{ek} = \mathbf{w}_0$.
- $\mathsf{Gen}(\mathsf{hk}, \mathbf{d}, \mathbf{s})$: Output

$$
\mathbf{C} = (\mathbf{I}_{t/n} \otimes \mathbf{s}^{\intercal}) \cdot \mathbf{U} + \mathbf{E}, \qquad \mathbf{h} = (\mathbf{I}_m \otimes \mathbf{s}^{\intercal}) \cdot \mathbf{d} + \mathbf{e},
$$

  where $\mathbf{E} \xleftarrow{\$} \mathcal{D}_{\bar{\sigma}}^{t/n \times t}$ and $\mathbf{e} \xleftarrow{\$} \mathcal{D}_{\sigma'}^{m}$.

- $\mathsf{Expand}(\mathsf{hk}, \mathsf{ek}, \mathbf{C}, \mathbf{h})$: Set $\mathbf{C}_0 = \mathbf{C}$. Then, inductively compute

$$\mathbf{C}_{i+1} = \mathbf{I}_c \otimes \mathbf{C}_i + (\mathbf{I}_c \otimes \mathbf{B}_i^\intercal) \cdot \mathbf{C} \cdot (\mathbf{I}_c \otimes \mathbf{L}_i).$$

  Output $\mathbf{v} = \mathbf{C}_{s-1} \cdot \mathbf{w}_0 + \mathbf{B}_{s-1}^\intercal \cdot \mathbf{h}$.

Before proving the correctness of expansion, we show the following useful technical lemma.

**Claim 1** (Noise Bound)**.** *For every $i \in [s]$, with overwhelming probability, it holds that*

$$\|\mathbf{C}_i - (\mathbf{I}_{m \cdot c^{i+1}} \otimes \mathbf{s}^\intercal) \cdot \mathbf{U}_i\|_\infty \leq 2\overline{\sigma}(tmn^2\sigma\sqrt{\lambda})^{2(i+1)}$$

*Proof.* We proceed by induction over $i$. It is easy to see that the base case $i = 0$ holds. For the inductive step, define $\mathbf{E}_i = \mathbf{C}_i - (\mathbf{I}_{m \cdot c^{i+1}} \otimes \mathbf{s}^\intercal) \cdot \mathbf{U}_i$. We observe that

$$\begin{aligned}
&\mathbf{C}_{i+1} - \mathbf{I}_c \otimes \mathbf{E}_i - (\mathbf{I}_c \otimes \mathbf{B}_i^\intercal) \cdot \mathbf{E} \cdot (\mathbf{I}_c \otimes \mathbf{L}_i) \\
&= \mathbf{I}_c \otimes \mathbf{C}_i + (\mathbf{I}_c \otimes \mathbf{B}_i^\intercal) \cdot \mathbf{C} \cdot (\mathbf{I}_c \otimes \mathbf{L}_i) - \mathbf{I}_c \otimes \mathbf{E}_i - (\mathbf{I}_c \otimes \mathbf{B}_i^\intercal) \cdot \mathbf{E} \cdot (\mathbf{I}_c \otimes \mathbf{L}_i) \\
&= \mathbf{I}_c \otimes (\mathbf{C}_i - \mathbf{E}_i) + (\mathbf{I}_c \otimes \mathbf{B}_i^\intercal) \cdot (\mathbf{C} - \mathbf{E}) \cdot (\mathbf{I}_c \otimes \mathbf{L}_i) \\
&= \mathbf{I}_c \otimes \Big((\mathbf{I}_{m \cdot c^{i+1}} \otimes \mathbf{s}^\intercal) \cdot \mathbf{U}_i\Big) + (\mathbf{I}_c \otimes \mathbf{B}_i^\intercal) \cdot \Big((\mathbf{I}_{m \cdot c} \otimes \mathbf{s}^\intercal) \cdot \mathbf{U}\Big) \cdot (\mathbf{I}_c \otimes \mathbf{L}_i) \\
&= (\mathbf{I}_{m \cdot c^{i+2}} \otimes \mathbf{s}^\intercal) \cdot (\mathbf{I}_c \otimes \mathbf{U}_i) + (\mathbf{I}_c \otimes \mathbf{B}_i^\intercal) \cdot (\mathbf{I}_{m \cdot c} \otimes \mathbf{s}^\intercal) \cdot \mathbf{U} \cdot (\mathbf{I}_c \otimes \mathbf{L}_i) \\
&= (\mathbf{I}_{m \cdot c^{i+2}} \otimes \mathbf{s}^\intercal) \cdot (\mathbf{I}_c \otimes \mathbf{U}_i) + (\mathbf{I}_c \otimes \mathbf{B}_i^\intercal \otimes \mathbf{s}^\intercal) \cdot \mathbf{U} \cdot (\mathbf{I}_c \otimes \mathbf{L}_i) \\
&= (\mathbf{I}_{m \cdot c^{i+2}} \otimes \mathbf{s}^\intercal) \cdot (\mathbf{I}_c \otimes \mathbf{U}_i) + (\mathbf{I}_{m \cdot c^{i+2}} \otimes \mathbf{s}^\intercal) \cdot (\mathbf{I}_c \otimes \mathbf{B}_i^\intercal \otimes \mathbf{I}_k) \cdot \mathbf{U} \cdot (\mathbf{I}_c \otimes \mathbf{L}_i) \\
&= (\mathbf{I}_{m \cdot c^{i+2}} \otimes \mathbf{s}^\intercal) \cdot \Big(\mathbf{I}_c \otimes \mathbf{U}_i + (\mathbf{I}_c \otimes \mathbf{B}_i^\intercal \otimes \mathbf{I}_k) \cdot \mathbf{U} \cdot (\mathbf{I}_c \otimes \mathbf{L}_i)\Big) \\
&= (\mathbf{I}_{m \cdot c^{i+2}} \otimes \mathbf{s}^\intercal) \cdot \mathbf{U}_{i+1},
\end{aligned}$$

so

$$\mathbf{E}_{i+1} = \mathbf{I}_c \otimes \mathbf{E}_i + (\mathbf{I}_c \otimes \mathbf{B}_i^\intercal) \cdot \mathbf{E} \cdot (\mathbf{I}_c \otimes \mathbf{L}_i).$$

With overwhelming probability, we have

$$\|(\mathbf{I}_c \otimes \mathbf{B}_i^\intercal) \cdot \mathbf{E} \cdot (\mathbf{I}_c \otimes \mathbf{L}_i)\|_\infty \leq m^2 n\overline{\sigma}\sqrt{\lambda} \cdot \|\mathbf{B}_i\|_\infty \cdot \|\mathbf{L}_i\|_\infty \leq \overline{\sigma}(tmn^2\sigma\sqrt{\lambda})^{2(i+2)}$$
$$\|\mathbf{I}_c \otimes \mathbf{E}_i\|_\infty \leq 2\overline{\sigma}(tmn^2\sigma\sqrt{\lambda})^{2(i+1)}.$$

Putting together these two, we obtain the claim. □

Next, we show that the output of the expansion algorithm is indeed a valid LWE sample with respect to the matrix $\mathbf{X}$.

**Lemma 17** (Correctness of Expansion)**.** *For all matrices $\mathbf{X} \in \mathbb{Z}_q^{k \times mc^s}$ it holds that*

$$\|\mathbf{s}^\intercal\mathbf{X} - \mathsf{Expand}(\mathsf{hk}, \mathsf{ek}, \mathbf{C}, \mathbf{h})^\intercal\|_\infty \leq 2tc^{s-1} \cdot \overline{\sigma}(tmn^2\sigma\sqrt{\lambda})^{2s} \cdot \|\mathsf{ek}\|_\infty + m(\sigma\sqrt{\lambda}tn^2m)^s \cdot \|\mathbf{e}\|_\infty$$

*with overwhelming probability, where $\mathsf{hk} \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$, $(\mathbf{d}, \mathsf{ek}) = \mathsf{Hash}(\mathsf{hk}, \mathbf{X}, \mathbf{w}')$, $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^k$, $\mathbf{E} \xleftarrow{\$} \mathcal{D}_{\overline{\sigma}}^{t/n \times t}$, $\mathbf{C} = (\mathbf{I}_{t/n} \otimes \mathbf{s}^\intercal) \cdot \mathbf{U} + \mathbf{E}$, and $\mathbf{h} = (\mathbf{I}_m \otimes \mathbf{s}^\intercal) \cdot \mathbf{d} + \mathbf{e}$. The probability is taken over the randomness of $\mathsf{Setup}$ and $\mathbf{E}$ only.*

*Proof.* Let $\mathbf{E}_i = \mathbf{C}_i - (\mathbf{I}_{m\cdot c^{i+1}} \otimes \mathbf{s}^\intercal) \cdot \mathbf{U}_i$. We expand

$$\begin{aligned}
\mathbf{v} &= \mathbf{C}_{s-1} \cdot \mathbf{w}_0 + \mathbf{B}_{s-1}^\intercal \cdot \mathbf{h} \\
&= \Big((\mathbf{I}_{m\cdot c^s} \otimes \mathbf{s}^\intercal) \cdot \mathbf{U}_{s-1} + \mathbf{E}_{s-1}\Big) \cdot \mathbf{w}_0 + \mathbf{B}_{s-1}^\intercal \cdot \Big((\mathbf{I}_m \otimes \mathbf{s}^\intercal) \cdot \mathbf{d} + \mathbf{e}\Big) \\
&= (\mathbf{I}_{m\cdot c^s} \otimes \mathbf{s}^\intercal) \cdot \mathbf{U}_{s-1} \cdot \mathbf{w}_0 + \mathbf{B}_{s-1}^\intercal \cdot (\mathbf{I}_m \otimes \mathbf{s}^\intercal) \cdot \mathbf{d} + \mathbf{E}_{s-1} \cdot \mathbf{w}_0 + \mathbf{B}_{s-1}^\intercal \cdot \mathbf{e} \\
&= (\mathbf{I}_{m\cdot c^s} \otimes \mathbf{s}^\intercal) \cdot \mathbf{U}_{s-1} \cdot \mathbf{w}_0 + (\mathbf{B}_{s-1}^\intercal \otimes \mathbf{s}^\intercal) \cdot \mathbf{d} + \mathbf{E}_{s-1} \cdot \mathbf{w}_0 + \mathbf{B}_{s-1}^\intercal \cdot \mathbf{e} \\
&= (\mathbf{I}_{m\cdot c^s} \otimes \mathbf{s}^\intercal) \cdot \mathbf{U}_{s-1} \cdot \mathbf{w}_0 + (\mathbf{B}_{s-1}^\intercal \otimes \mathbf{s}^\intercal) \cdot (\mathbf{I}_m \otimes \mathbf{G}) \cdot \mathbf{w}_1 + \mathbf{E}_{s-1} \cdot \mathbf{w}_0 + \mathbf{B}_{s-1}^\intercal \cdot \mathbf{e} \\
&= (\mathbf{I}_{m\cdot c^s} \otimes \mathbf{s}^\intercal) \cdot \mathbf{U}_{s-1} \cdot \mathbf{w}_0 + (\mathbf{I}_{m\cdot c^s} \otimes \mathbf{s}^\intercal) \cdot (\mathbf{B}_{s-1}^\intercal \otimes \mathbf{G}) \cdot \mathbf{w}_1 + \mathbf{E}_{s-1} \cdot \mathbf{w}_0 + \mathbf{B}_{s-1}^\intercal \cdot \mathbf{e} \\
&= (\mathbf{I}_{m\cdot c^s} \otimes \mathbf{s}^\intercal) \cdot \Big(\mathbf{U}_{s-1} \cdot \mathbf{w}_0 + (\mathbf{B}_{s-1}^\intercal \otimes \mathbf{G}) \cdot \mathbf{w}_1\Big) + \mathbf{E}_{s-1} \cdot \mathbf{w}_0 + \mathbf{B}_{s-1}^\intercal \cdot \mathbf{e} \\
&= (\mathbf{I}_{m\cdot c^s} \otimes \mathbf{s}^\intercal) \cdot \mathbf{x} + \mathbf{E}_{s-1} \cdot \mathbf{w}_0 + \mathbf{B}_{s-1}^\intercal \cdot \mathbf{e} \\
&= (\mathbf{I}_{m\cdot c^s} \otimes \mathbf{s}^\intercal) \cdot \mathsf{vec}(\mathbf{X}) + \mathbf{E}_{s-1} \cdot \mathbf{w}_0 + \mathbf{B}_{s-1}^\intercal \cdot \mathbf{e} \\
&= \mathsf{vec}(\mathbf{s}^\intercal \mathbf{X}) + \mathbf{E}_{s-1} \cdot \mathbf{w}_0 + \mathbf{B}_{s-1}^\intercal \cdot \mathbf{e} \\
&= \mathbf{X}^\intercal \mathbf{s} + \mathbf{E}_{s-1} \cdot \mathbf{w}_0 + \mathbf{B}_{s-1}^\intercal \cdot \mathbf{e}.
\end{aligned}$$

By Claim 1, we have

$$\|\mathbf{E}_{s-1} \cdot \mathbf{w}_0 + \mathbf{B}_{s-1}^\intercal \cdot \mathbf{e}\|_\infty \leq 2tc^{s-1} \cdot \overline{\sigma}(tmn^2\sigma\sqrt{\lambda})^{2s} \cdot \|\mathbf{w}_0\|_\infty + m(\sigma\sqrt{\lambda}tn^2m)^s \cdot \|\mathbf{e}\|_\infty$$

concluding our proof. □

### 5.2 Derandomized Indistinguishability

We state and prove the security notion that is satisfied by our construction. In order to do this, we first have to introduce some additional notation. Fix an arbitrary PPT algorithm $\mathsf{Extract}$ of depth at most $D$. We rely on three separate instances of the input-succinct laconic shift-hiding PRF (Section 4): $\mathsf{SH}, \mathsf{SH}_1, \mathsf{SH}_2$. Each instance has its own internal lattice modulus and output modulus. Below, the output modulus of $\mathsf{SH}$ is the sampler modulus $q$, while the output moduli of $\mathsf{SH}_1$ and $\mathsf{SH}_2$ are $p$; the corresponding internal lattice moduli are separate, larger values satisfying the condition in Section 4.2. We define the three functions as follows:

- ($\mathsf{SH}$) Defined with respect to a public function

$$\begin{aligned}
f(\alpha, \beta) &= f((i, \mathbf{Y}, \mathsf{aux}), (\tilde{K}_1, K_2)) \\
&= \Big[-(\mathbf{B}_{s-1}^\intercal \otimes \mathbf{G}) \cdot \mathsf{SH}_1.\mathsf{ShiftEval}(\tilde{K}_1, (\mathbf{Y}, \mathsf{aux})) + \mathbf{Y} \cdot \mathsf{SH}_2.\mathsf{Eval}(K_2, (\mathbf{Y}, \mathsf{aux}))\Big]_i \in \mathbb{Z}_q
\end{aligned}$$

  where $[\cdot]_i$ denotes the $i$-th entry. Above $i$ ranges over $[kmc^s]$.

- ($\mathsf{SH}_1$) Defined with respect to a public function

$$\begin{aligned}
f_1(\alpha, \beta) &= f_1((i, \mathbf{Y}, \mathsf{aux}), (K, \mathbf{M}_0, \mathbf{M}_1)) \\
&= \big[\mathbf{L}_{s-1} \cdot \mathbf{G}^{-1}(\mathsf{SH.Eval}(K, (\mathbf{Y}, \mathsf{aux}))) - \mathbf{G}^{-1}(\mathsf{vec}(\mathbf{M}_0 \mathbf{K}_{\mathbf{M}_0, \mathbf{Y}} \mathbf{V} + \mathbf{M}_1))\big]_i \in \mathbb{Z}_p
\end{aligned}$$

  where $\mathbf{K}_{\mathbf{M}_0, \mathbf{Y}} = \mathsf{BGG.EvalM}(\mathbf{M}_0, \mathsf{SH}_1.\mathsf{Eval}(\cdot, (\mathbf{Y}, \mathsf{aux})))$ as defined in [BGG+14]. Moreover, we assume that $\mathsf{SH}_1$ is $(mnt^s + 1)$-intractable. Above $i$ ranges over $[nm]$.

- ($\mathsf{SH}_2$) Defined with respect to a public function

$$f_2(\alpha,\beta) = f_2((i,\mathbf{Y},\mathsf{aux}),(\tilde{K},\tau)) \\ = \left[-\mathsf{Extract}(\tau,\mathbf{Y},\mathsf{aux})\cdot\mathbf{G}^{-1}(\mathsf{SH.ShiftEval}(\tilde{K},(\mathbf{Y},\mathsf{aux})))\right]_i \in \mathbb{Z}_p.$$

  Above $i$ ranges over the columns of the matrix $\mathbf{Y}$. We further assume that $\mathsf{SH}_2$ is $mnc^s$-intractable.

From now on, when we need to evaluate a shifted or unshifted key on an input $(i,\mathbf{Y},\mathsf{aux})$ for every index $i$ and then stack the results to form a vector, we abuse notation and we simply evaluate it on $(\mathbf{Y},\mathsf{aux})$. Observe that despite the circular dependency, the three schemes $\mathsf{SH}$, $\mathsf{SH}_1$ and $\mathsf{SH}_2$ are still well defined by their laconic properties. This is because our shift hiding function supports any polynomial sized Boolean circuit of bounded depth $\delta$. Since the depth of the evaluation of *unshifted* keys is $\mathsf{poly}(\log\delta,\log\lambda)$, we can find a choice of parameters for which the above construction is well-defined. In particular, if $\mathsf{SH}$, $\mathsf{SH}_1$ and $\mathsf{SH}_2$ support computations of depth at most $\delta$, $\delta_1$ and $\delta_2$, we require that $\delta_2 \geq D+\delta\cdot\mathsf{poly}(\log\lambda)$, $\delta \geq \delta_1\cdot\mathsf{poly}(\log\lambda)$ and $\delta_1 \geq (\log(\delta_1)+\log(\delta))\cdot\mathsf{poly}\log(\lambda)$, where $D$ is the depth of $\mathsf{Extract}$. We also require that

$$p = t^s\cdot 2^{\omega(\log\lambda)} \qquad t = cnm \qquad \overline{\sigma} = 2^{\omega(\log\lambda)} \qquad \sigma' = \overline{\sigma}\cdot 2^{\log\delta_1\cdot\omega(\log\lambda)}$$

where $c>1$. Let $\sigma_{\mathrm{LWE}}$ be the challenge-noise parameter of the decomposed LWE assumption, and require $\overline{\sigma}/(n\sigma_{\mathrm{LWE}}\sqrt{\lambda}) = 2^{\omega(\log\lambda)}$. We are now ready to formulate the security definition for our compressed sampler. We mention that here we prove a slightly more general definition than what is actually needed for our application, specifically we include an auxiliary input to the PRF, which we imagine could be a useful generalization for future applications of this theorem.

The shift-input lengths of $\mathsf{SH}_1,\mathsf{SH},\mathsf{SH}_2$ are $\eta_1,\eta,\eta_2$, respectively. Choose polynomial bounds satisfying

$$\eta_1 \geq |(K,\mathbf{M}_0,\mathbf{M}_1)|, \qquad \eta \geq |(\tilde{K}_1,K_2)|, \qquad \eta_2 \geq |(\tilde{K},\tau)|.$$

Here lengths are binary representation lengths, maximized over the generated keys and the prescribed inputs. All shift inputs are padded with zeros to the corresponding length.

**Definition 5** (Derandomized Indistinguishability)**.** *Consider the following experiment between a PPT attacker and a challenger.*

- *(Preliminary Phase) The attacker sends $\tau$ to the challenger.*
- *(Setup Phase) The challenger samples*

$$\mathbf{D} \xleftarrow{\$} \mathbb{Z}_q^{k\times m} \qquad \mathsf{hk} \xleftarrow{\$} \mathsf{Setup}(1^\lambda) \qquad K_1 \xleftarrow{\$} \mathsf{SH}_1.\mathsf{Setup}(1^\lambda,1^{\eta_1}) \qquad K_2 \xleftarrow{\$} \mathsf{SH}_2.\mathsf{Setup}(1^\lambda,1^{\eta_2})$$

  *and sets $\tilde{K}_1 \xleftarrow{\$} \mathsf{SH}_1.\mathsf{Shift}(K_1,0)$ $\tilde{K}_2 \xleftarrow{\$} \mathsf{SH}_2.\mathsf{Shift}(K_2,0)$. Send $(\mathbf{D},\mathsf{hk},\tilde{K}_1,\tilde{K}_2)$ to the attacker.*
- *(Admissibility) The attacker returns a pair $(\mathbf{Y},\mathsf{aux})$ and the challenger executes $\mathbf{T} = \mathsf{Extract}(\tau,\mathbf{Y},\mathsf{aux})$ and checks if*

$$\|\mathbf{T}\|_\infty \leq 1 \qquad \textit{and} \qquad \mathbf{Y}\cdot\mathbf{T} = \mathbf{I}_{mc^s}\otimes\mathbf{G}.$$

  *The challenger aborts if any of these checks fail; on abort, the attacker wins with probability $1/2$, independently of its view.*

- *(Challenge Phase) The challenger samples a random bit* $b \xleftarrow{\$} \{0,1\}$ *and proceeds as follows.*
  - *If* $b = 1$*: Sample* $\mathbf{h} \xleftarrow{\$} \mathbb{Z}_q^m$, $\mathbf{C} \xleftarrow{\$} \mathbb{Z}_q^{t/n \times t}$, *and* $\mathbf{a} \xleftarrow{\$} \mathbb{Z}_q^m$.
  - *If* $b = 0$*: Compute*

$$\begin{aligned} \mathbf{r}_1 &= \mathsf{SH}_1.\mathsf{ShiftEval}(\tilde{K}_1, (\mathbf{Y}, \mathsf{aux})) \\ \mathbf{u} &= \mathsf{SH}_2.\mathsf{ShiftEval}(\tilde{K}_2, (\mathbf{Y}, \mathsf{aux})) \\ (\mathbf{d}, \mathsf{ek}) &= \mathsf{Hash}\left(\mathsf{hk}, \mathbf{X}, \begin{bmatrix} \mathbf{0}^{t \cdot c^{s-1}} \\ \mathbf{r}_1 \end{bmatrix}\right) \end{aligned}$$

    *where* $\mathbf{X} \in \mathbb{Z}_q^{k \times mc^s}$ *is such that* $\mathsf{vec}(\mathbf{X}) = \mathbf{Yu}$. *Sample* $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^k$ *and* $\mathbf{e} \xleftarrow{\$} \mathcal{D}_{\bar{\sigma}}^m$ *then set* $(\mathbf{C}, \mathbf{h}) \xleftarrow{\$} \mathsf{Gen}(\mathsf{hk}, \mathbf{d}, \mathbf{s})$ *and* $\mathbf{a} = \mathbf{s}^\intercal \mathbf{D} + \mathbf{e}^\intercal$.

- *(Answer Phase) The attacker is given* $(\mathbf{C}, \mathbf{h}, \mathbf{a})$ *and returns a bit* $b'$. *The attacker wins if* $b' = b$.

*We say that a compressed sampler satisfies derandomized indistinguishability if, for every PPT algorithm* Extract *of the prescribed dimensions and depth bound and every PPT attacker, the probability that the attacker wins in the above experiment is negligibly close to* $\frac{1}{2}$.

Observe that with our choice of derandomization, we have that $\|\mathsf{ek}\|_\infty \leq 1$. We are now ready to state the security of our construction.

**Theorem 8** (Derandomized Indistinguishability)**.** *Assuming the hardness of the decomposed LWE problem with challenge-noise parameter* $\sigma_{\mathrm{LWE}}$*, the construction as defined above satisfies derandomized indistinguishability.*

*Proof.* To prove the result, we proceed by means of a sequence of computationally indistinguishable hybrids.

**Hybrid 0:** This corresponds to the game in which $b = 0$.

**Hybrid 1:** In this hybrid, we change the distribution of $\tilde{K}_2$. We sample $K \xleftarrow{\$} \mathsf{SH.Setup}(1^\lambda, 1^\eta)$ and we set $\tilde{K} \xleftarrow{\$} \mathsf{SH.Shift}(K, 0)$. Then, we set $\tilde{K}_2 \xleftarrow{\$} \mathsf{SH}_2.\mathsf{Shift}(K_2, (\tilde{K}, \tau))$. Observe that this hybrid is indistinguishable from Hybrid 0 by the shift hiding property of $\mathsf{SH}_2$.

**Hybrid 2:** In this hybrid, we change the distribution of $\tilde{K}$. We set $\tilde{K} \xleftarrow{\$} \mathsf{SH.Shift}(K, (\tilde{K}_1, K_2))$. Observe that this hybrid is indistinguishable from Hybrid 1 by the shift-hiding property of $\mathsf{SH}$.

**Hybrid 3:** In this hybrid, we change the distribution of $\tilde{K}_1$. We sample random matrices $\mathbf{M}_0 \xleftarrow{\$} \mathbb{Z}_q^{k \times n\ell_1}$ and $\mathbf{M}_1 \xleftarrow{\$} \mathbb{Z}_q^{k \times m}$, where $\ell_1$ is the bit-length of $K_1$. Then, we sample $\tilde{K}_1 \xleftarrow{\$} \mathsf{SH}_1.\mathsf{Shift}(K_1, (K, \mathbf{M}_0, \mathbf{M}_1))$. Observe that this hybrid is indistinguishable from Hybrid 2 by the shift-hiding property of $\mathsf{SH}_1$.

**Hybrid 4:** In this hybrid, we change the way we compute $\mathbf{d}$. Specifically, we set it to

$$\mathbf{d} = (\mathbf{I}_m \otimes \mathbf{G}) \cdot \mathsf{SH}_1.\mathsf{Eval}(K_1, (\mathbf{Y}, \mathsf{aux})) - \mathsf{vec}(\mathbf{M}_0 \cdot \mathbf{K}_{\mathbf{M}_0,\mathbf{Y}} \cdot \mathbf{V} + \mathbf{M}_1).$$

Recall that, in Hybrid 3, $\mathbf{d}$ is defined as

$$\mathbf{d} = (\mathbf{I}_m \otimes \mathbf{G}) \cdot \mathbf{w}_1 = (\mathbf{I}_m \otimes \mathbf{G}) \cdot (\mathbf{r}_1 - \mathbf{L}_{s-1} \cdot \mathbf{G}^{-1}(\mathbf{t}))$$

where $\mathbf{t} = (\mathbf{B}_{s-1}^\intercal \otimes \mathbf{G}) \cdot \mathbf{r}_1 - \mathbf{Y} \cdot \mathbf{u}$. By the correctness of the shift-hiding function, we have that

$$\mathbf{r}_1 = \mathsf{SH}_1.\mathsf{Eval}(K_1, (\mathbf{Y}, \mathsf{aux})) + \mathbf{L}_{s-1} \cdot \mathbf{G}^{-1}(\overline{\mathbf{t}}) - \mathbf{G}^{-1}(\mathsf{vec}(\mathbf{M}_0 \cdot \mathbf{K}_{\mathbf{M}_0,\mathbf{Y}} \cdot \mathbf{V} + \mathbf{M}_1)) \tag{32}$$

with $\overline{\mathbf{t}} = \mathsf{SH}.\mathsf{Eval}(K, (\mathbf{Y}, \mathsf{aux}))$. In fact Equation (32) holds over the integers (without modular reduction) since $\mathsf{SH}_1.\mathsf{Eval}(K_1, \cdot)$ is $(mnt^s + 1)$-intractable and

$$\|\mathbf{L}_{s-1} \cdot \mathbf{G}^{-1}(\overline{\mathbf{t}}) - \mathbf{G}^{-1}(\mathsf{vec}(\mathbf{M}_0 \cdot \mathbf{K}_{\mathbf{M}_0,\mathbf{Y}} \cdot \mathbf{V} + \mathbf{M}_1))\|_\infty \leq mnt^s + 1.$$

Moreover, by the correctness of $\mathsf{SH}_2$, we have that

$$\mathbf{u} = \overline{\mathbf{u}} - \mathbf{T} \cdot \mathbf{G}^{-1}(\tilde{\mathbf{t}}) \tag{33}$$

where $\tilde{\mathbf{t}} = \mathsf{SH}.\mathsf{ShiftEval}(\tilde{K}, (\mathbf{Y}, \mathsf{aux}))$ and $\overline{\mathbf{u}} = \mathsf{SH}_2.\mathsf{Eval}(K_2, (\mathbf{Y}, \mathsf{aux}))$. Again, Equation (33) holds over the integers (no modular reduction) since $\mathsf{SH}_2.\mathsf{Eval}(K_2, \cdot)$ is $mnc^s$-intractable and

$$\|\mathbf{T} \cdot \mathbf{G}^{-1}(\tilde{\mathbf{t}})\|_\infty \leq mnc^s.$$

Finally, by the correctness of $\mathsf{SH}$, we have that

$$\tilde{\mathbf{t}} = \overline{\mathbf{t}} - (\mathbf{B}_{s-1}^\intercal \otimes \mathbf{G}) \cdot \mathbf{r}_1 + \mathbf{Y} \cdot \overline{\mathbf{u}}.$$

Then we can use Equation (33) to conclude that

$$\begin{aligned}
\overline{\mathbf{t}} &= \tilde{\mathbf{t}} + (\mathbf{B}_{s-1}^\intercal \otimes \mathbf{G}) \cdot \mathbf{r}_1 - \mathbf{Y} \cdot \overline{\mathbf{u}} \\
&= \tilde{\mathbf{t}} + (\mathbf{B}_{s-1}^\intercal \otimes \mathbf{G}) \cdot \mathbf{r}_1 - \mathbf{Y} \cdot \mathbf{u} - \mathbf{Y} \cdot \mathbf{T} \cdot \mathbf{G}^{-1}(\tilde{\mathbf{t}}) \\
&= \tilde{\mathbf{t}} + (\mathbf{B}_{s-1}^\intercal \otimes \mathbf{G}) \cdot \mathbf{r}_1 - \mathbf{Y} \cdot \mathbf{u} - (\mathbf{I}_{m \cdot c^s} \otimes \mathbf{G}) \cdot \mathbf{G}^{-1}(\tilde{\mathbf{t}}) \\
&= \tilde{\mathbf{t}} + (\mathbf{B}_{s-1}^\intercal \otimes \mathbf{G}) \cdot \mathbf{r}_1 - \mathbf{Y} \cdot \mathbf{u} - \tilde{\mathbf{t}} \\
&= (\mathbf{B}_{s-1}^\intercal \otimes \mathbf{G}) \cdot \mathbf{r}_1 - \mathbf{Y} \cdot \mathbf{u} \\
&= \mathbf{t}.
\end{aligned}$$

Substituting in Equation (32), we obtain

$$\mathbf{d} = (\mathbf{I}_m \otimes \mathbf{G}) \cdot \mathbf{w}_1 = (\mathbf{I}_m \otimes \mathbf{G}) \cdot \mathsf{SH}_1.\mathsf{Eval}(K_1, (\mathbf{Y}, \mathsf{aux})) - \mathsf{vec}(\mathbf{M}_0 \cdot \mathbf{K}_{\mathbf{M}_0,\mathbf{Y}} \cdot \mathbf{V} + \mathbf{M}_1)$$

which is precisely defined as in Hybrid 4.

**Hybrid 5:** In this hybrid, we change again how we compute $\mathbf{d}$. We set it to

$$\mathbf{d} = -\mathsf{vec}\left((\mathbf{M}_0 - \mathsf{Bits}(K_1)^\intercal \otimes \mathbf{G}) \cdot \mathbf{H}_{\mathbf{M}_0,\mathbf{Y},K_1} \cdot \mathbf{V} + \mathbf{M}_1\right),$$

where $\mathbf{H}_{\mathbf{M}_0,\mathbf{Y},K_1} := \mathsf{BGG.EvalC}(\mathbf{M}_0, K_1, \mathsf{SH}_1.\mathsf{Eval}(\cdot, (\mathbf{Y}, \mathsf{aux})))$. This hybrid is identical to Hybrid 4 by the key equation of BGG encodings [BGG+14]. Indeed,

$$\begin{aligned}
&(\mathbf{M}_0 - \mathsf{Bits}(K_1)^\intercal \otimes \mathbf{G}) \cdot \mathbf{H}_{\mathbf{M}_0,\mathbf{Y},K_1} \cdot \mathbf{V} \\
&\quad = (\mathbf{M}_0 \cdot \mathbf{K}_{\mathbf{M}_0,\mathbf{Y}} - \mathsf{SH}_1.\mathsf{Eval}(K_1, (\mathbf{Y}, \mathsf{aux}))^\intercal \otimes \mathbf{G}) \cdot \mathbf{V} \\
&\quad = \mathbf{M}_0 \cdot \mathbf{K}_{\mathbf{M}_0,\mathbf{Y}} \cdot \mathbf{V} - (\mathsf{SH}_1.\mathsf{Eval}(K_1, (\mathbf{Y}, \mathsf{aux}))^\intercal \otimes \mathbf{G}) \cdot \mathbf{V}.
\end{aligned}$$

Moreover, for any vector $\mathbf{v} \in \mathbb{Z}_q^{nm}$, we have that $\mathsf{vec}((\mathbf{v}^\intercal \otimes \mathbf{G}) \cdot \mathbf{V}) = (\mathbf{I}_m \otimes \mathbf{G}) \cdot \mathbf{v}$.

**Hybrid 6:** In this hybrid, we change the distribution of $\mathbf{M}_0$. Specifically, we sample a random $\mathbf{Q}$ and we set

$$\mathbf{M}_0 = \mathbf{Q} + \mathsf{Bits}(K_1)^\intercal \otimes \mathbf{G},$$

where $\mathsf{Bits}(K_1)$ is the representation of $K_1$ as a binary string. Moreover, we compute $\mathbf{d}$ as follows

$$\mathbf{d} = -\mathsf{vec}(\mathbf{Q} \cdot \mathbf{H}_{\mathbf{M}_0,\mathbf{Y},K_1} \cdot \mathbf{V} + \mathbf{M}_1).$$

This hybrid is perfectly indistinguishable from Hybrid 5.

**Hybrid 7:** In this hybrid, we change the way we compute $\mathbf{h}$. We set it to

$$\mathbf{h} = -(\mathbf{v}_0^\intercal \cdot \mathbf{H}_{\mathbf{M}_0,\mathbf{Y},K_1} \cdot \mathbf{V})^\intercal - \mathbf{v}_1 + \mathbf{e}_2$$

where

$$\mathbf{e}_0 \xleftarrow{\$} \mathcal{D}_{\overline{\sigma}}^{n\ell_1} \qquad \mathbf{e}_1 \xleftarrow{\$} \mathcal{D}_{\overline{\sigma}}^{m} \qquad \mathbf{e}_2 \xleftarrow{\$} \mathcal{D}_{\sigma'}^{m} \qquad \mathbf{v}_0^\intercal = \mathbf{s}^\intercal \cdot \mathbf{Q} + \mathbf{e}_0^\intercal \qquad \mathbf{v}_1^\intercal = \mathbf{s}^\intercal \cdot \mathbf{M}_1 + \mathbf{e}_1^\intercal.$$

Since $\sigma' = \overline{\sigma} \cdot 2^{\log \delta_1 \cdot \omega(\log \lambda)}$, this hybrid is statistically indistinguishable from Hybrid 6 by a smudging argument (Lemma 4).

**Hybrid 8:** In this hybrid, we sample $\mathbf{v}_0 \xleftarrow{\$} \mathbb{Z}_q^{n\ell_1}$, $\mathbf{a} \xleftarrow{\$} \mathbb{Z}_q^m$, $\mathbf{v}_1 \xleftarrow{\$} \mathbb{Z}_q^m$, and $\mathbf{C} \xleftarrow{\$} \mathbb{Z}_q^{t/n \times t}$. This hybrid is computationally indistinguishable from Hybrid 7 under the hardness of decomposed LWE. Indeed, by Equation (29), we have

$$\begin{aligned}
\mathbf{C}^\intercal &= \mathbf{U}^\intercal \cdot (\mathbf{I}_{t/n} \otimes \mathbf{s}) + \mathbf{E}^\intercal \\
&= (\mathbf{I}_t \otimes \mathbf{s}^\intercal) \cdot \Pi_k(\mathbf{U}) + \mathbf{E}^\intercal \\
&= (\mathbf{I}_t \otimes \mathbf{s}^\intercal) \cdot \Pi_k(\Pi_k(\mathbf{AB}) + \mathbf{I}_{t/n} \otimes \mathbf{G}) + \mathbf{E}^\intercal \\
&= (\mathbf{I}_t \otimes \mathbf{s}^\intercal) \cdot (\mathbf{AB} + \Pi_k(\mathbf{I}_{t/n} \otimes \mathbf{G})) + \mathbf{E}^\intercal \\
&= (\mathbf{I}_t \otimes \mathbf{s}^\intercal) \cdot (\mathbf{A}\tilde{\mathbf{B}} + \mathbf{I}_t \otimes \mathbf{G}) \cdot \mathbf{G}^{-1}(\Pi_k(\mathbf{I}_{t/n} \otimes \mathbf{G})) + \mathbf{E}^\intercal.
\end{aligned}$$

Write $\mathbf{S} = \mathbf{G}^{-1}(\Pi_k(\mathbf{I}_{t/n} \otimes \mathbf{G}))$. Use Definition 1 with the auxiliary random-matrix samples described after that definition, with challenge-noise parameter $\sigma_{\mathrm{LWE}}$ and auxiliary matrix $[\mathbf{Q} \mid \mathbf{M}_1 \mid \mathbf{D}]$. Multiply the decomposed sample by $\mathbf{S}$, and add independent $\mathcal{D}_{\overline{\sigma}}$ noise to every entry of this product and of the three auxiliary samples. Each column of $\mathbf{S}$ has $n$ ones, and its column supports are disjoint. Thus each transformed challenge-noise entry has magnitude at most $n\sigma_{\mathrm{LWE}}\sqrt{\lambda}$ with overwhelming probability. The new parameter condition and Lemma 4 make this joint distribution statistically close to Hybrid 7. In the uniform case, multiplication by $\mathbf{S}$ is surjective, so these samples are jointly uniform, as in Hybrid 8.

**Hybrid 9:** In this hybrid, we sample $\mathbf{h} \xleftarrow{\$} \mathbb{Z}_q^m$. This hybrid is perfectly indistinguishable from Hybrid 8.

**Hybrid 10:** In this hybrid, we set $\tilde{K}_1 \xleftarrow{\$} \mathsf{SH}_1.\mathsf{Shift}(K_1, 0)$ and $\tilde{K}_2 \xleftarrow{\$} \mathsf{SH}_2.\mathsf{Shift}(K_2, 0)$. This hybrid is indistinguishable from Hybrid 9 by the shift-hiding properties of $\mathsf{SH}$, $\mathsf{SH}_1$ and $\mathsf{SH}_2$ (to see it, reverse Hybrid 3, Hybrid 2 and Hybrid 1). Notice that at this point the game is identical to the case $b = 1$.

This ends the proof. □

# 6 Broadcast Encryption

In the following we show our main result, namely a construction of adaptively secure (distributed) broadcast encryption.

## 6.1 Definition

Broadcast encryption [FN94] allows one to broadcast a message to a set of $N$ participants. The most basic notion [FN94] considers a central key generation algorithm that provides keys to all participants. *Distributed* broadcast [BZ14, KMW23] allows each party to sample their own public-secret key pair. More recently, *flexible* broadcast [FWW23] further extends this notion to remove parties' identifiers. Here we show a formal definition for the latter.

**Syntax.** We recall the syntax of a flexible broadcast scheme.

- $\mathsf{Setup}(1^\lambda)$: On input the security parameter $1^\lambda$, the setup algorithm returns a master public key $\mathsf{mpk}$.
- $\mathsf{KeyGen}(\mathsf{mpk})$: On input the master public key $\mathsf{mpk}$, the key generation algorithm returns a public-secret key pair $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{Enc}(\mathsf{mpk}, \mathsf{pk}_0, \ldots, \mathsf{pk}_{N-1}, \mathsf{msg})$: On input the master public key $\mathsf{mpk}$, a set of public keys $\mathsf{pk}_0, \ldots, \mathsf{pk}_{N-1}$, and a message $\mathsf{msg} \in \{0,1\}$, the encryption algorithm returns a ciphertext $\mathsf{ct}$.
- $\mathsf{Dec}(\mathsf{mpk}, k, \mathsf{sk}, \mathsf{ct}, \mathsf{pk}_0, \ldots, \mathsf{pk}_{N-1})$: On input the master public key $\mathsf{mpk}$, the index $k$, the secret key $\mathsf{sk}$, a ciphertext $\mathsf{ct}$, and the public keys $\mathsf{pk}_0, \ldots, \mathsf{pk}_{N-1}$, the decryption algorithm returns the message $\mathsf{msg}$.

For *correctness*, we require that for all $k \in [N]$ we have

$$\mathsf{Dec}(\mathsf{mpk}, k, \mathsf{sk}_k, \mathsf{Enc}(\mathsf{mpk}, \mathsf{pk}_0, \ldots, \mathsf{pk}_{N-1}, \mathsf{msg}), \mathsf{pk}_0, \ldots, \mathsf{pk}_{N-1}) = \mathsf{msg}$$

where $(\mathsf{pk}_k, \mathsf{sk}_k) \xleftarrow{\$} \mathsf{KeyGen}(\mathsf{mpk})$ and $\mathsf{mpk} \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$, except with negligible probability. For *efficiency*, we require that the ciphertext size should be bounded by a function that is (at most) poly-logarithmic in the number of users $N$.

**Security.** We define the following version of security, referred to as *semi-static* security [GW09, FWW23], as an experiment where the attacker is allowed to select an arbitrary set of honestly-sampled keys.

**Definition 6** (Semi-Static Security)**.** *Consider the following experiment between a PPT attacker and a challenger.*

- *(Setup Phase) The challenger samples the master public key* $\mathsf{mpk} \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$ *and sends it to the attacker.*
- *(Query Phase) The attacker can issue polynomially-many queries to the challenger, who executes* $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(\mathsf{mpk})$ *and returns* $\mathsf{pk}$ *to the attacker.*
- *(Challenge Phase) The attacker outputs a set of keys* $\mathsf{pk}_0, \ldots, \mathsf{pk}_{N-1}$ *and the challenger checks, for all* $k \in [N]$*, that* $\mathsf{pk}_k$ *is part of the keys sent in the query phase. The challenger aborts if this is not the case. On an abort, the attacker is declared to win with probability* $\frac{1}{2}$*.*

  *Otherwise, the challenger samples a random bit* $b \xleftarrow{\$} \{0,1\}$ *and returns*

  $$\mathsf{ct} \xleftarrow{\$} \mathsf{Enc}(\mathsf{mpk}, \mathsf{pk}_0, \ldots, \mathsf{pk}_{N-1}, b)$$

  *to the attacker.*
- *(Output Phase) The attacker returns a bit* $b'$ *and wins if* $b = b'$*.*

*We say that a flexible broadcast is semi-statically secure if for all PPT attackers the probability that they win in the above experiment is negligibly close to* $\frac{1}{2}$*.*

A stronger notion of security, called *adaptive* security, is defined as above except that the attacker is allowed to corrupt keys sampled in the query phase, and receive the corresponding secret key. The challenger should additionally check that the challenge set does not contain any corrupted public key. In [GW09] it is shown that any flexible broadcast satisfying semi-static security can be turned into one satisfying adaptive security in the random oracle model. In [HWW25] they show a standard model transformation for the case of distributed broadcast. Since flexible broadcast implies distributed broadcast, the following allows us to concentrate on the notion of semi-static security.

**Theorem 9** ([HWW25])**.** *Assuming the hardness of the LWE problem and a distributed broadcast satisfying semi-static security, then there exists a distributed broadcast satisfying adaptive security.*

### 6.2 Our Scheme

We are now ready to formally present our scheme. Let $\mathsf{Comp}$ denote the already derandomized succinct LWE sampler, i.e., we generate the randomness for the hashing algorithm using $\mathsf{SH}_1, \mathsf{SH}_2$ as in Definition 5. Write each public key as $\mathsf{pk}_i = (\xi_i, \mathbf{Q}_i)$. Here $\mathsf{aux}$ is the ordered list of recipient tags $\xi_i$, and every PRF evaluation below on $\mathbf{Y}$ means evaluation on $(\mathbf{Y}, \mathsf{aux})$. We choose the recursion depth $s$ so that $N \le mc^s$ and pad every recipient list to $mc^s$ blocks using the fixed dummy public key $(\perp, \mathbf{G})$; hence, below we assume without loss of generality that $N = mc^s$. For this application, instantiate $\mathsf{SH}_2$ with output length $n + m$: its shift $f_2$ returns the corresponding recipient block of

the vector in Section 5.2. A scalar coordinate $j$ means coordinate $j$ mod $(n+m)$ of the evaluation at block index $\lfloor j/(n+m) \rfloor$; all stacked evaluations use this convention. Let $q$ be a modulus such that

$$q \geq 2^{\omega(\log \lambda)} \left( (\overline{\sigma} + \sigma' \sqrt{\lambda})(ctmn^2 \sigma \sqrt{\lambda})^{2s+1} + m\overline{\sigma}\sqrt{\lambda} n^2 p \right)$$

where $p$ is the modulus used in $\mathsf{SH}_1$ and $\mathsf{SH}_2$.

- $\mathsf{Setup}(1^\lambda)$: Sample $K_1 \xleftarrow{\$} \mathsf{SH}_1.\mathsf{Setup}(1^\lambda, 1^{\eta_1})$ and $K_2 \xleftarrow{\$} \mathsf{SH}_2.\mathsf{Setup}(1^\lambda, 1^{\eta_2})$ and compute

  $$\mathsf{hk} \xleftarrow{\$} \mathsf{Comp.Setup}(1^\lambda) \quad \tilde{K}_1 \xleftarrow{\$} \mathsf{SH}_1.\mathsf{Shift}(K_1, 0) \quad \tilde{K}_2 \xleftarrow{\$} \mathsf{SH}_2.\mathsf{Shift}(K_2, 0) \quad \mathbf{D} \xleftarrow{\$} \mathbb{Z}_q^{k \times m}$$

  and output the master public key $\mathsf{mpk} := (\mathsf{hk}, \tilde{K}_1, \tilde{K}_2, \mathbf{D})$.

- $\mathsf{KeyGen}(\mathsf{mpk})$: Compute

  $$\mathbf{Q} = \mathbf{D} \cdot \mathbf{R}$$

  where $\mathbf{R} \xleftarrow{\$} \mathbb{Z}_2^{m \times n}$ and $\xi \xleftarrow{\$} \{0,1\}^\lambda$ independently. Output $\mathsf{pk} = (\xi, \mathbf{Q})$ and $\mathsf{sk} = \mathbf{R}$.

- $\mathsf{Enc}(\mathsf{mpk}, \mathsf{pk}_0, \ldots, \mathsf{pk}_{N-1}, \mathsf{msg})$: Define the matrix

  $$\mathbf{Y} = \begin{bmatrix} \mathbf{Q}_0 \mid \mathbf{D} & & \\ & \ddots & \\ & & \mathbf{Q}_{N-1} \mid \mathbf{D} \end{bmatrix}$$

  and compute

  $$\mathbf{r}_1 = \mathsf{SH}_1.\mathsf{ShiftEval}(\tilde{K}_1, \mathbf{Y}) \qquad \text{and} \qquad \mathbf{u} = \mathsf{SH}_2.\mathsf{ShiftEval}(\tilde{K}_2, \mathbf{Y}).$$

  Then set

  $$\mathbf{c}^\mathsf{T} = \mathbf{s}^\mathsf{T}\mathbf{D} + \mathbf{e}^\mathsf{T} + \mathsf{msg} \cdot \mathbf{t}^\mathsf{T} \qquad \text{and} \qquad \tilde{\mathbf{r}} = \begin{bmatrix} \mathbf{0} \\ \mathbf{r}_1 \end{bmatrix}$$

  where $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^k$, $\mathbf{t} \xleftarrow{\$} \mathbb{Z}_q^m$, and $\mathbf{e} \xleftarrow{\$} \mathcal{D}_{\overline{\sigma}}^m$. Derive the matrix $\mathbf{X} \in \mathbb{Z}_q^{k \times N}$ such that $\mathsf{vec}(\mathbf{X}) = \mathbf{Y} \cdot \mathbf{u}$ and compute

  $$(\mathbf{d}, \mathsf{ek}) = \mathsf{Comp.Hash}(\mathsf{hk}, \mathbf{X}, \tilde{\mathbf{r}}) \qquad (\mathbf{C}, \mathbf{h}) \xleftarrow{\$} \mathsf{Comp.Gen}(\mathsf{hk}, \mathbf{d}, \mathbf{s}).$$

  Output $\mathsf{ct} = (\mathbf{C}, \mathbf{h}, \mathbf{c})$.

- $\mathsf{Dec}(\mathsf{mpk}, i, \mathsf{sk}, \mathsf{ct}, \mathsf{pk}_0, \ldots, \mathsf{pk}_{N-1})$: Recompute $(\mathbf{d}, \mathsf{ek})$ as in the $\mathsf{Enc}$ algorithm, which are deterministically derived from the public keys, then expand

  $$\mathbf{v} = \mathsf{Comp.Expand}(\mathsf{hk}, \mathsf{ek}, \mathbf{C}, \mathbf{h}).$$

  Let $\mathbf{v}_i$ denote the $i$-th entry of $\mathbf{v}$. Parse

  $$\mathbf{u} = \begin{bmatrix} \mathbf{u}_0 \\ \vdots \\ \mathbf{u}_{N-1} \end{bmatrix}, \qquad \mathbf{u}_i = \begin{bmatrix} \mathbf{u}_i^{(0)} \\ \mathbf{u}_i^{(1)} \end{bmatrix}$$

where $\mathbf{u}_i^{(0)} \in \mathbb{Z}_q^n$ and $\mathbf{u}_i^{(1)} \in \mathbb{Z}_q^m$. Finally, return 0 if

$$\left\| \mathbf{v}_i - \mathbf{c}^\intercal \mathbf{R}_i \mathbf{u}_i^{(0)} - \mathbf{c}^\intercal \mathbf{u}_i^{(1)} \right\|_\infty \leq 3(\overline{\sigma} + \sigma' \sqrt{\lambda})(ctmn^2 \sigma \sqrt{\lambda})^{2s+1} + m\overline{\sigma}\sqrt{\lambda} \cdot (n+1) \cdot p$$

and 1 otherwise.

To see that the scheme satisfies correctness, observe that, by Lemma 17, we can rewrite

$$\begin{aligned} \mathbf{v}_i &= \mathbf{s}^\intercal \cdot \left[ \begin{array}{c|c} \mathbf{Q}_i & \mathbf{D} \end{array} \right] \cdot \mathbf{u}_i + \hat{\mathbf{e}}^\intercal \\ &= \mathbf{s}^\intercal \mathbf{Q}_i \mathbf{u}_i^{(0)} + \mathbf{s}^\intercal \mathbf{D} \mathbf{u}_i^{(1)} + \hat{\mathbf{e}}^\intercal \\ &= \mathbf{s}^\intercal \mathbf{D} \mathbf{R}_i \mathbf{u}_i^{(0)} + \mathbf{s}^\intercal \mathbf{D} \mathbf{u}_i^{(1)} + \hat{\mathbf{e}}^\intercal. \end{aligned}$$

Substituting the definition of $\mathbf{c}$, we can see that

$$\begin{aligned} \mathbf{v}_i - \mathbf{c}^\intercal \mathbf{R}_i \mathbf{u}_i^{(0)} - \mathbf{c}^\intercal \mathbf{u}_i^{(1)} &= \mathbf{v}_i - \mathbf{s}^\intercal \mathbf{D} \mathbf{R}_i \mathbf{u}_i^{(0)} - \mathbf{s}^\intercal \mathbf{D} \mathbf{u}_i^{(1)} - \mathbf{e}^\intercal (\mathbf{R}_i \mathbf{u}_i^{(0)} + \mathbf{u}_i^{(1)}) - \mathsf{msg} \cdot \mathbf{t}^\intercal (\mathbf{R}_i \mathbf{u}_i^{(0)} + \mathbf{u}_i^{(1)}) \\ &= \hat{\mathbf{e}}^\intercal - \mathbf{e}^\intercal (\mathbf{R}_i \mathbf{u}_i^{(0)} + \mathbf{u}_i^{(1)}) - \mathsf{msg} \cdot \mathbf{t}^\intercal (\mathbf{R}_i \mathbf{u}_i^{(0)} + \mathbf{u}_i^{(1)}). \end{aligned}$$

If $\mathsf{msg} = 0$, then it suffices to bound

$$\left\| \hat{\mathbf{e}}^\intercal - \mathbf{e}^\intercal (\mathbf{R}_i \mathbf{u}_i^{(0)} + \mathbf{u}_i^{(1)}) \right\|_\infty \leq 3(\overline{\sigma} + \sigma' \sqrt{\lambda})(ctmn^2 \sigma \sqrt{\lambda})^{2s+1} + m\overline{\sigma}\sqrt{\lambda} \cdot (n+1) \cdot p.$$

On the other hand, if $\mathsf{msg} = 1$, the mask contribution is $\mathbf{t}^\intercal \mathbf{w}_i$, where $\mathbf{w}_i := \mathbf{R}_i \mathbf{u}_i^{(0)} + \mathbf{u}_i^{(1)}$. Conditioned on $\mathbf{w}_i$ containing a unit of $\mathbb{Z}_q$, this inner product is uniform over $\mathbb{Z}_q$. Here $q$ is a power of two, so a residue is a unit exactly when it is odd. For honestly generated recipient keys, condition on $\mathbf{R}_i$ and $\mathbf{u}_i^{(0)}$; the independent uniform output-mask coordinates in $\mathbf{u}_i^{(1)}$ make the coordinates of $\mathbf{w}_i$ independently uniform in parity. Thus the probability that none is a unit is $2^{-m}$. For large enough $q$, decryption therefore outputs 1 with probability $> 0.9$. The probability of an incorrect decryption can be driven to negligible by encrypting the message multiple times and taking a majority vote.

For efficiency, it is easy to see that the sizes of the ciphertext, of the public/secret key pairs, and of the master public key are bounded by a function poly-logarithmic in the number of users $N$. Before proving security, we state a lemma that is going to be useful for our analysis.

**Lemma 18** (Key Indistinguishability)**.** *The following distributions are statistically close*

$$\left\{ \mathbf{D}, \mathbf{DR} : \mathbf{D} \xleftarrow{\$} \mathbb{Z}_q^{k \times m}, \mathbf{R} \xleftarrow{\$} \mathbb{Z}_2^{m \times n} \right\} \approx_s \left\{ \mathbf{D}, \mathbf{DR} + \mathbf{G} : \mathbf{D} \xleftarrow{\$} \mathbb{Z}_q^{k \times m}, \mathbf{R} \xleftarrow{\$} \mathbb{Z}_2^{m \times n} \right\}.$$

*Proof.* Follows as a direct consequence of Lemma 1. □

We are now ready to state our main theorem of this section.

**Theorem 10** (Semi-Static Security)**.** *If the decomposed LWE assumption holds, the scheme as described above satisfies semi-static security (Definition 6).*

*Proof.* By Theorem 8 it suffices to show an admissible adversary, i.e., such that we can extract a valid trapdoor $\mathbf{T}_i$ for all matrices $\left[ \begin{array}{c|c} \mathbf{Q}_i & \mathbf{D} \end{array} \right]$ with $i \in [N]$. Then, by Definition 5, the ciphertext is computationally indistinguishable from uniform.

First, let us consider a modified experiment where, for all key queries, we sample keys as $\mathbf{Q}_i = \mathbf{D}\mathbf{R}_i + \mathbf{G}$. By Lemma 18, the view of this modified experiment is statistically close to the original one. Note that in this modified experiment it holds that

$$\left[\ \mathbf{Q}_i \ \middle|\ \mathbf{D}\ \right] \cdot \underbrace{\begin{bmatrix} \mathbf{I}_n \\ -\mathbf{R}_i \end{bmatrix}}_{\mathbf{T}_i} = \mathbf{G}$$

with $\|\mathbf{T}_i\|_\infty \leq 1$. Thus, $\mathbf{T}_i$ is a valid trapdoor. Second, we modify the experiment, and we sample each $\mathbf{R}_i$ using a regular $\mathsf{PRF}(\tau, \xi_i)$, for a random $\tau$. Except for the negligible probability of a tag collision, this change is indistinguishable by the pseudorandomness of the PRF.

Given $(\tau, \mathbf{Y}, \mathsf{aux})$, let $\mathsf{Extract}$ return the block-diagonal matrix with blocks $[\mathbf{I}_n; -\mathsf{PRF}(\tau, \xi_i)]$ for recipient tags and $[\mathbf{I}_n; \mathbf{0}]$ for dummy tags $\perp$, returning zero on malformed inputs. This polynomial-time extractor uses only the compact seed $\tau$ and produces a valid trapdoor for every challenge set in the modified experiment, making the adversary admissible. This ends the proof. □

# 7 Pseudorandom Obfuscation

An obfuscation scheme [BGI+01] consists of a PPT algorithm $\mathsf{Obf}$ that outputs an obfuscated version of an input function $\tilde{f} \xleftarrow{\$} \mathsf{Obf}(1^\lambda, f)$, such that for any input $x$ it holds that

$$\tilde{f}(x) = f(x).$$

The standard notion of security for an obfuscation scheme is indistinguishability obfuscation (iO) [BGI+01].

**Definition 7** (Indistinguishability Obfuscation)**.** *An obfuscation scheme* $\mathsf{Obf}$ *satisfies indistinguishability obfuscation if, for all functionally equivalent* $f_0 \equiv f_1$*, it holds that*

$$\left\{\tilde{f} : \tilde{f} \xleftarrow{\$} \mathsf{Obf}(1^\lambda, f_0)\right\} \approx_c \left\{\tilde{f} : \tilde{f} \xleftarrow{\$} \mathsf{Obf}(1^\lambda, f_1)\right\}.$$

More recently [BDJ+25] put forward the notion of *pseudorandom obfuscation* for keyed functions. The security of the primitive postulates that the obfuscation itself should be computationally indistinguishable from uniform. We take the function output length $\mu$ and the canonical obfuscation length $L_{\mathsf{Obf}}$ to be fixed polynomials in the security parameter.

**Definition 8** (Pseudorandom Obfuscation)**.** *An obfuscation scheme* $\mathsf{Obf}$ *satisfies pseudorandomness if, for all functions* $f_k$ *such that*

$$\left\{f_k(x) : k \xleftarrow{\$} \{0,1\}^\lambda\right\}_x \approx_c \left\{u_x : u_x \xleftarrow{\$} \{0,1\}^\mu\right\}_x$$

*it holds that*

$$\left\{\tilde{f} : \tilde{f} \xleftarrow{\$} \mathsf{Obf}(1^\lambda, f_k), k \xleftarrow{\$} \{0,1\}^\lambda\right\} \approx_c \left\{U : U \xleftarrow{\$} \{0,1\}^{L_{\mathsf{Obf}}}\right\}.$$

In [BDJ+25] it is shown that such a notion is subject to strong impossibility results for general functions $f$. Here we consider a special class of functions, that we refer to as *puncturable* functions

[SW14]. We say that a keyed function is puncturable if there exists an algorithm $\mathsf{Puncture}$ such that for all $k \in \{0,1\}^\lambda$ and all $x \neq x^*$ it holds that

$$f_k(x) = f_{k_{x^*}}(x)$$

where $k \xleftarrow{\$} \{0,1\}^\lambda$ and $k_{x^*} = \mathsf{Puncture}(k, x^*)$. Furthermore, for all $x^* \in \{0,1\}^d$ we require that

$$\left\{ f_k(x^*), \mathsf{Puncture}(k, x^*) : k \xleftarrow{\$} \{0,1\}^\lambda \right\} \approx_c \left\{ u, \mathsf{Puncture}(k, x^*) : k \xleftarrow{\$} \{0,1\}^\lambda, u \xleftarrow{\$} \{0,1\}^\mu \right\}.$$

In this section we show how to turn an iO for puncturable PRFs into a pseudorandom one, assuming our laconic shift-hiding PRF[3] from Section 4. For notational convenience, we are going to assume that the shift algorithm inputs a function instead of a string. We describe the scheme in the following.

- $\mathsf{PRObf}(1^\lambda, f_k)$: Let $\eta$ be a fixed bound on the encoding length of the shift description, sample a key $K \xleftarrow{\$} \mathsf{Setup}(1^\lambda, 1^\eta)$, and define

  $$g_{K,k}(x) = f_k(x) - \mathsf{Eval}(K, x).$$

  Sample $\tilde{g} \xleftarrow{\$} \mathsf{Obf}(1^\lambda, g_{K,k})$ and return $K_{\tilde{g}} = \mathsf{Shift}(K, \tilde{g})$.

We claim that evaluating the shifted key returns the correct output. This can be easily established by observing that, by Lemma 13, we have

$$\mathsf{ShiftEval}(K_{\tilde{g}}, x) = \mathsf{Eval}(K, x) + \tilde{g}(x) = \mathsf{Eval}(K, x) + g_{K,k}(x) = f_k(x)$$

except with negligible probability. Note that in order for the construction to be well-defined, we require that the circuit depth of the shifted function $\widetilde{g}$ should be bounded by a fixed polynomial in the security parameter (since the complexity of the shift-hiding PRF depends on the *depth* of the shifted function). This can be achieved by combining our construction from Section 4 with an iO where the depth of the evaluation circuit grows polynomially with the *depth* of the underlying function. In turn, this can be generically achieved using standard techniques [BDJ+25] (e.g., obfuscating a circuit that returns a randomized encoding of the output). All that is left to be shown is that the construction described above satisfies pseudorandomness.

**Theorem 11** (Pseudorandomness)**.** *Suppose decomposed LWE,* $\mathsf{Obf}$*, and the punctured-point pseudorandomness of* $f_k$ *have security* $2^{-\Omega(\lambda^\epsilon)}$ *against polynomial-time adversaries for some* $\epsilon > 0$*, and* $d = o(\lambda^\epsilon)$*. Then the scheme satisfies pseudorandomness (Definition 8) for all puncturable functions with the stated parameters.*

*Proof.* We prove security via a sequence of hybrid experiments, where we gradually change the distribution of $\mathsf{PRObf}(1^\lambda, f_k)$. For each $x^* \in \{0,1\}^d$, in lexicographical order we define the following hybrids.

[3]Here we assume that the parameters $p$ and $q$ as defined in Section 4 are powers of 2 so that uniform elements of $\mathbb{Z}_p$ and $\mathbb{Z}_q$ can be canonically mapped to uniform bitstrings. This is purely for notational convenience, and it is straightforward to extend the definition and construction to different moduli. We also assume a canonical invertible map from discrete Gaussians to random strings, and we refer the reader to [BDJ+25] for a precise description.

**Hybrid 0:** We obfuscate the function $g_{K,k}$ defined as

$$g_{K,k}(x) := f_k(x) - \mathbf{1}_{x \geq x^*} \mathsf{Eval}(K, x).$$

For the first $x^*$ this is the original distribution, and otherwise it is identical to Hybrid 5 of the preceding cycle.

**Hybrid 1:** We compute the punctured key $k_{x^*} = \mathsf{Puncture}(k, x^*)$ and obfuscate the function $g'_{K,k}$ defined as

$$g'_{K,k}(x) = \begin{cases} g_{K,k_{x^*}}(x) & \text{if } x \neq x^* \\ f_k(x^*) - \mathsf{Eval}(K, x^*) & \text{otherwise.} \end{cases}$$

Note that this function is equivalent to the previous hybrid ($g'_{K,k} \equiv g_{K,k}$), so indistinguishability follows by Definition 7.

**Hybrid 2:** We sample a random string $u$ and we define

$$g'_{K,k_{x^*},u}(x) = \begin{cases} g_{K,k_{x^*}}(x) & \text{if } x \neq x^* \\ u - \mathsf{Eval}(K, x^*) & \text{otherwise.} \end{cases}$$

This change is computationally indistinguishable by the puncturable pseudorandomness of $f_k$.

**Hybrid 3:** We redefine

$$g'_{K,k_{x^*},u}(x) = \begin{cases} g_{K,k_{x^*}}(x) & \text{if } x \neq x^* \\ u & \text{otherwise.} \end{cases}$$

Since $u$ is uniform, this change is only syntactical.

**Hybrid 4:** We revert the modification done in the second hybrid and set

$$g'_{K,k}(x) = \begin{cases} g_{K,k_{x^*}}(x) & \text{if } x \neq x^* \\ f_k(x^*) & \text{otherwise.} \end{cases}$$

Indistinguishability follows with the same argument.

**Hybrid 5:** We revert the modification done in the first hybrid and set

$$g'_{K,k}(x) = \begin{cases} g_{K,k}(x) & \text{if } x \neq x^* \\ f_k(x^*) & \text{otherwise.} \end{cases}$$

Indistinguishability follows with the same argument.

Observe that, after all the hybrids (for all $x \in \{0,1\}^d$, in lexicographical order), the final-hybrid shifted function is $f_k$, which is independent of $K$; consequently, so is its obfuscation $\tilde{g}$. Thus, by the pseudorandomness of the shift-hiding PRF (refer to Lemma 14), $\mathsf{Shift}(K, \tilde{g}) = \mathsf{PRObf}(1^\lambda, f_k)$ is computationally close to uniform. The total loss is a factor of at most $5 \cdot 2^d$, which is absorbed by the quantitative assumptions above. □

### AI Disclosure

ChatGPT was used solely for writing assistance. The authors take full responsibility for all claims and proofs made in this paper.

### Acknowledgments

We thank Jeffrey Champion for pointing out a bug in an early version of this work and Hoeteck Wee for suggestions that improved the presentation of this work. The constructions and proofs were produced by humans.

D.A. was supported by a fellowship with the Simons Institute for the Theory of Computing. This work was supported by Input Output (iog.io) through their funding of the Edinburgh Blockchain Technology Lab. G.M. is supported by the European Research Council through an ERC Starting Grant (Grant agreement No. 101077455, ObfusQation) and by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA – 390781972. L.R. was supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement number 101124977 (DECRYPSIS).

Part of this work was conducted while the authors were visiting the Simons Institute for the Theory of Computing.

## References

[AJ15] Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 308–326. Springer, Berlin, Heidelberg, August 2015.

[AJSW26] Siddharth Agarwal, Abhishek Jain, Akshayaram Srinivasan, and David Wu. Simultaneous-message and succinct secure computation: Reusable and multiparty protocols. In *EUROCRYPT 2026*, 2026.

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems (extended abstract). In *28th ACM STOC*, pages 99–108. ACM Press, May 1996.

[AMR25a] Damiano Abram, Giulio Malavolta, and Lawrence Roy. Key-homomorphic computations for RAM: Fully succinct randomised encodings and more. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part III*, volume 16002 of *LNCS*, pages 236–268. Springer, Cham, August 2025.

[AMR25b] Damiano Abram, Giulio Malavolta, and Lawrence Roy. Succinct oblivious tensor evaluation and applications: Adaptively-secure laconic function evaluation and trapdoor hashing for all circuits. In Michal Koucký and Nikhil Bansal, editors, *57th ACM STOC*, pages 1875–1886. ACM Press, June 2025.

[AMR26] Damiano Abram, Giulio Malavolta, and Lawrence Roy. Tree encodings ii: Correlation intractability for all batched relations. Cryptology ePrint Archive, Paper 2026/1140, 2026.

[Ban93] Wojciech Banaszczyk. New bounds in some transference theorems in the geometry of numbers. *Mathematische Annalen*, 296:625–635, 1993.

[BC12] Nir Bitansky and Alessandro Chiesa. Succinct arguments from multi-prover interactive proofs and their efficiency benefits. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 255–272. Springer, Berlin, Heidelberg, August 2012.

[BDGM22] Zvika Brakerski, Nico Döttling, Sanjam Garg, and Giulio Malavolta. Factoring and pairings are not necessary for IO: Circular-secure LWE suffices. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *ICALP 2022*, volume 229 of *LIPIcs*, pages 28:1–28:20. Schloss Dagstuhl, July 2022.

[BDJ+25] Pedro Branco, Nico Döttling, Abhishek Jain, Giulio Malavolta, Surya Mathialagan, Spencer Peters, and Vinod Vaikuntanathan. Pseudorandom obfuscation and applications. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part V*, volume 16004 of *LNCS*, pages 663–698. Springer, Cham, August 2025.

[BGG+14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 533–556. Springer, Berlin, Heidelberg, May 2014.

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 1–18. Springer, Berlin, Heidelberg, August 2001.

[BLW17] Dan Boneh, Kevin Lewi, and David J. Wu. Constraining pseudorandom functions privately. In Serge Fehr, editor, *PKC 2017, Part II*, volume 10175 of *LNCS*, pages 494–524. Springer, Berlin, Heidelberg, March 2017.

[BÜW24] Chris Brzuska, Akin Ünal, and Ivy K. Y. Woo. Evasive LWE assumptions: Definitions, classes, and counterexamples. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part IV*, volume 15487 of *LNCS*, pages 418–449. Springer, Singapore, December 2024.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In Rafail Ostrovsky, editor, *52nd FOCS*, pages 97–106. IEEE Computer Society Press, October 2011.

[BV15a] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. In Venkatesan Guruswami, editor, *56th FOCS*, pages 171–190. IEEE Computer Society Press, October 2015.

[BV15b] Zvika Brakerski and Vinod Vaikuntanathan. Constrained key-homomorphic PRFs from standard lattice assumptions - or: How to secretly embed a circuit in your PRF. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 1–30. Springer, Berlin, Heidelberg, March 2015.

[BW06] Dan Boneh and Brent Waters. A fully collusion resistant broadcast, trace, and revoke system. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006*, pages 211–220. ACM Press, October / November 2006.

[BZ14] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 480–499. Springer, Berlin, Heidelberg, August 2014.

[CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited (preliminary version). In *30th ACM STOC*, pages 209–218. ACM Press, May 1998.

[CHW25] Jeffrey Champion, Yao-Ching Hsieh, and David J. Wu. Registered ABE and adaptively-secure broadcast encryption from succinct LWE. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part III*, volume 16002 of *LNCS*, pages 3–34. Springer, Cham, August 2025.

[CJJ22] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. SNARGs for $\mathcal{P}$ from LWE. In *62nd FOCS*, pages 68–79. IEEE Computer Society Press, February 2022.

[DJM+25] Nico Döttling, Abhishek Jain, Giulio Malavolta, Surya Mathialagan, and Vinod Vaikuntanathan. Simple and general counterexamples for private-coin evasive LWE. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part VII*, volume 16006 of *LNCS*, pages 73–92. Springer, Cham, August 2025.

[FN94] Amos Fiat and Moni Naor. Broadcast encryption. In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 480–491. Springer, Berlin, Heidelberg, August 1994.

[FWW23] Cody Freitag, Brent Waters, and David J. Wu. How to use (plain) witness encryption: Registered ABE, flexible broadcast, and more. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part IV*, volume 14084 of *LNCS*, pages 498–531. Springer, Cham, August 2023.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009.

[GKW18] Rishab Goyal, Venkata Koppula, and Brent Waters. Collusion resistant traitor tracing from learning with errors. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *50th ACM STOC*, pages 660–670. ACM Press, June 2018.

[GP21] Romain Gay and Rafael Pass. Indistinguishability obfuscation from circular security. In Samir Khuller and Virginia Vassilevska Williams, editors, *53rd ACM STOC*, pages 736–749. ACM Press, June 2021.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 75–92. Springer, Berlin, Heidelberg, August 2013.

[GVW13] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Attribute-based encryption for circuits. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 545–554. ACM Press, June 2013.

[GW09] Craig Gentry and Brent Waters. Adaptive security in broadcast encryption systems (with short ciphertexts). In Antoine Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 171–188. Springer, Berlin, Heidelberg, April 2009.

[GY26a] Rishab Goyal and Saikumar Yadugiri. Adaptively-secure flexible and identity-based broadcast encryption from decomposed LWE. Cryptology ePrint Archive, Paper 2026/862, 2026.

[GY26b] Rishab Goyal and Saikumar Yadugiri. Equivocal broadcast encryption: Adaptively-secure optimal distributed broadcast encryption from lattices. In *CRYPTO 2026*, 2026.

[HWW25] Yao-Ching Hsieh, Brent Waters, and David J. Wu. A generic approach to adaptively-secure broadcast encryption in the plain model. In Serge Fehr and Pierre-Alain Fouque, editors, *EUROCRYPT 2025, Part III*, volume 15603 of *LNCS*, pages 336–365. Springer, Cham, May 2025.

[IKO07] Yuval Ishai, Eyal Kushilevitz, and Rafail Ostrovsky. Efficient arguments without short pcps. In *Twenty-Second Annual IEEE Conference on Computational Complexity (CCC'07)*, pages 278–291, 2007.

[ILL89] Russell Impagliazzo, Leonid A. Levin, and Michael Luby. Pseudo-random generation from one-way functions (extended abstracts). In *21st ACM STOC*, pages 12–24. ACM Press, May 1989.

[KMW23] Dimitris Kolonelos, Giulio Malavolta, and Hoeteck Wee. Distributed broadcast encryption from bilinear groups. In Jian Guo and Ron Steinfeld, editors, *ASIACRYPT 2023, Part V*, volume 14442 of *LNCS*, pages 407–441. Springer, Singapore, December 2023.

[LRY16] Benoît Libert, Somindu C. Ramanna, and Moti Yung. Functional commitment schemes: From polynomial commitments to pairing-based accumulators from simple assumptions. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *ICALP 2016*, volume 55 of *LIPIcs*, pages 30:1–30:14. Schloss Dagstuhl, July 2016.

[LV22] Alex Lombardi and Vinod Vaikuntanathan. Correlation-intractable hash functions via shift-hiding. In Mark Braverman, editor, *ITCS 2022*, volume 215, pages 102:1–102:16. LIPIcs, January / February 2022.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 700–718. Springer, Berlin, Heidelberg, April 2012.

[PS18] Chris Peikert and Sina Shiehian. Privately constraining and programming PRFs, the LWE way. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part II*, volume 10770 of *LNCS*, pages 675–701. Springer, Cham, March 2018.

[QWW18] Willy Quach, Hoeteck Wee, and Daniel Wichs. Laconic function evaluation and applications. In Mikkel Thorup, editor, *59th FOCS*, pages 859–870. IEEE Computer Society Press, October 2018.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *46th ACM STOC*, pages 475–484. ACM Press, May / June 2014.

[Tsa22] Rotem Tsabary. Candidate witness encryption from lattice techniques. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part I*, volume 13507 of *LNCS*, pages 535–559. Springer, Cham, August 2022.

[VWW22] Vinod Vaikuntanathan, Hoeteck Wee, and Daniel Wichs. Witness encryption and null-IO from evasive LWE. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part I*, volume 13791 of *LNCS*, pages 195–221. Springer, Cham, December 2022.

[Wat09] Brent Waters. Dual system encryption: Realizing fully secure IBE and HIBE under simple assumptions. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 619–636. Springer, Berlin, Heidelberg, August 2009.

[Wee24] Hoeteck Wee. Circuit ABE with $\mathsf{poly}(\mathsf{depth}, \lambda)$-sized ciphertexts and keys from lattices. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part III*, volume 14922 of *LNCS*, pages 178–209. Springer, Cham, August 2024.

[Wee25] Hoeteck Wee. Almost optimal KP and CP-ABE for circuits from succinct LWE. In Serge Fehr and Pierre-Alain Fouque, editors, *EUROCRYPT 2025, Part III*, volume 15603 of *LNCS*, pages 34–62. Springer, Cham, May 2025.

[WW21] Hoeteck Wee and Daniel Wichs. Candidate obfuscation via oblivious LWE sampling. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part III*, volume 12698 of *LNCS*, pages 127–156. Springer, Cham, October 2021.

[WW25] Hoeteck Wee and David J. Wu. Unbounded distributed broadcast encryption and registered ABE from succinct LWE. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part III*, volume 16002 of *LNCS*, pages 204–235. Springer, Cham, August 2025.