On Perfect Correctness in (Lockable) Obfuscation

Rishab Goyal* Venkata Koppula† Satyanarayana Vusirikala‡ Brent Waters§

September 9, 2019

Abstract

In a lockable obfuscation scheme [GKW17a, WZ17] a party takes as input a program P, a lock value α, a message msg and produces an obfuscated program $\widetilde{P}$. The obfuscated program can be evaluated on an input x to learn the message msg if $P(x) = \alpha$. The security of such schemes states that if α is randomly chosen (independent of P and msg), then one cannot distinguish an obfuscation of P from a "dummy" obfuscation. Existing constructions of lockable obfuscation achieve provable security under the Learning with Errors assumption. One limitation of these constructions is that they achieve only statistical correctness and allow for a possible one sided error where the obfuscated program could output the msg on some value x where $P(x) \neq \alpha$.

In this work we motivate the problem of studying perfect correctness in lockable obfuscation for the case where the party performing the obfuscation might wish to inject a backdoor or hole in correctness. We begin by studying the existing constructions and identify two components that are susceptible to imperfect correctness. The first is in the LWE-based pseudo random generators (PRGs) that are non-injective, while the second is in the last level testing procedure of the core constructions.

We address each in turn. First, we build upon previous work to design *injective* PRGs that are provably secure from the LWE assumption. Next, we design an alternative last level testing procedure that has additional structure to prevent correctness errors. We then provide a surgical proof of security (to avoid redundancy) that connects our construction to the construction by Goyal, Koppula, and Waters (GKW) [GKW17a]. Specifically, we show how for a random value α an obfuscation under our new construction is indistinguishable from an obfuscation under the existing GKW construction.

1 Introduction

In cryptographic program obfuscation a user wants to take a program P and publish an obfuscated program $\widetilde{P}$. The obfuscated program should maintain the same functionality of the original while intuitively hiding anything about the structure of P beyond what can be determined by querying its input/output functionality.

One issue in defining semantics is whether we demand that $\widetilde{P}$ always match the functionality exactly on all inputs or we relax correctness to allow for some deviation with negligible probability. At first blush such differences in semantics might appear to be very minor. With a negligible correctness error it is straightforward for the obfsucator to parameterize an obfuscation such that the probability of a correctness error is some minuscule value such as 2^{-300} which would be much less than say the probability of dying from an asteroid strike (1 in 74 million).

The idea that statistical correctness is always good enough, however, rests on the presumption that the obfuscator itself wants to avoid errors. Consider for example, an party that is tasked with building a program that screens images from a video feed and raises an alert if any suspicious activity is detected. The party

[*] University of Texas at Austin. Email: `rgoyal@cs.utexas.edu`. Supported by IBM PhD Fellowship.

[†] Weizmann Institute of Science. Email: `venkata.koppula@weizmann.ac.il`.

[‡] University of Texas at Austin. Email: `satya@cs.utexas.edu`.

[§] University of Texas at Austin. Email: `bwaters@cs.utexas.edu`. Supported by NSF CNS-1414082, DARPA SafeWare, Microsoft Faculty Fellowship, and Packard Foundation Fellowship.

could first create and program P to perform this function and then release an obfuscated version $\widetilde{P}$ that could hide features proprietary vision recognition algorithm) about how the program was built. But what if the party wants to abuse their role? For instance, they might want to publish a program $\widetilde{P}$ that unfairly flags a certain group or individual. Or perhaps is programmed with a backdoor to let a certain image pass.

In an obfuscation scheme with perfect correctness, it might be possible to audit such behavior. For example, an auditor could require that the obfuscating party produce their original program P along with the random coins used in obfuscating it. Then the auditor could check that the original program P meets certain requirements as well as seeing that $\widetilde{P}$ is indeed an obfuscation of P.[1] (We emphasize that if one does not want to reveal P to an auditor that such a proof can be done in zero knowledge or by attaching a non-interactive zero knowledge proof to the program.) However, for such a process to work it is imperative that the obfuscation algorithm be perfectly correct. Otherwise, a malicious obfuscator could potentially start with a perfectly legitimate program P, but purposefully choose coins that would flip the output of a program at a particular input point. In addition to the above scenario where obfuscation was its own application, perfect correctness might be required when obfuscation is used as a building block in other primitives. For example, in the work of Bitansky, Khurana, and Paneth [BKP19] they required lockable obfuscation with one-sided perfect correctness.[2]

In this paper we initiate a study on perfect correctness in obfuscation by studying it in lockable obfuscation, which is arguably the most powerful form of obfuscation which is provably secure under a standard assumption. Recall that a lockable obfuscation [GKW17a, WZ17] scheme takes as input a program P, a message msg, a lock value α and produces an obfuscated program $\widetilde{P}$. The semantics of evaluation are such that on input x the evaluation of the program outputs msg if and only if $P(x) = \alpha$. Lockable obfuscation security requires that the obfuscation of any program P with a randomly (and independently of P and msg) chosen value α will be indistinguishable from a "dummy" obfuscated program that is created without any knowledge of P and msg other than their sizes. While the power of lockable obfuscation does not reach that of indistinguishability obfuscation [BGI$^+$01, GGH$^+$13, SW14], it has been shown to be sufficient for many applications such as obfuscating conjunction and affine testers, upgrading public key encryption, identity-based encryption [Sha85, BF01, Coc01] and attribute-based encryption [SW05] to their anonymous versions and giving random oracle uninstantiatability and circular security separation results, and most recently, building efficient traitor tracing systems [BSW06, CVW$^+$18a].

The works of Goyal, Koppula, and Waters [GKW17a] and Wichs and Zirdelis [WZ17] introduced and gave constructions of lockable obfuscation provably secure under the Learning with Errors [Reg05] assumption. A limitation of both constructions (inherited from the bit-encryption cycle testers of [GKW17c]) is that they provide only statistical correctness. In particular, there exists a one-sided error in which it is possible that there exists an input x such that $P(x) \neq \alpha$ yet the obfuscated program outputs msg on input x.

Our Results. With this motivation in mind we seek to create a lockable obfuscation scheme that is perfectly correct and retains the provable security under the LWE assumption. We begin by examining the GKW lockable obfuscation for branching prorgrams and identify two points in the construction that are susceptible to correctness errors. The first is in the use of an LWE-base pseudo random generator that could be non-injective. The second is in the "last level testing procedure" comprised in the core construction. We address each one in turn. First, we build over the previous work to design and prove a new PRG construction that is both injective and probably secure from the LWE assumption. (We also create an injective PRG from the learning parity with noise (LPN) assumption as an added bonus.) Then we look to surgically modify the GKW construction to change the last level testing procedure to avoid the correctness pitfall. We accomplish this by adding more structure to a final level of matrices to avoid false matches, but doing so makes the new construction incompatible with the existing security proof. Instead of re-deriving the entire proof of security, we carefully show how an obfuscation under our new construction with a random lock value is indistinguishable from an obfuscation under the previous construction. Security then follows.

[1] The above argument relies on the ability of one being able to test the original program meets a certain template or is otherwise well-formed. Our work does not address under which circumstances this is possible.

[2] In this particular example perfect correctness [GKW17a, WZ17] was already present for the side they needed.

While the focus of this work has been on constructing lockable obfuscation schemes with perfect correctness building upon the schemes of [GKW17a, WZ17], we believe our techniques can also be applied to the recent obfuscation scheme by Chen, Vaikuntanathan, and Wee [CVW18b].

1.1 Technical Overview

We first present a short overview of the statistically correct lockable obfuscation scheme by Goyal et al. [GKW17b, Appendix D], (henceforth referred to as the GKW scheme), and discuss the barriers to achieving perfect correctness. Next, we discuss how to overcome each of these barriers in order to achieve perfect correctness.

Overview of the GKW scheme. The GKW scheme can be broken down into three parts: (i) constructing a lockable obfuscation scheme for NC^1 circuits and 1-bit messages, (ii) bootstrapping to lockable obfuscation for poly-depth circuits, and (iii) extending to multi-bit message space. It turns out that steps (ii) and (iii) preserve the correctness properties of the underlying lockable obfuscation scheme, thus in order to build a *perfectly correct* lockable obfuscation scheme for poly-depth circuits and multi-bit messages, we only need to build a *perfectly correct* lockable obfuscation scheme for NC^1 and 1-bit messages.[3] We start by giving a brief overview of the lockable obfuscation scheme for NC^1, and then move to highlight the barriers to achieving perfect correctness.

One of the key ingredients in the GKW construction is a family of log-depth (statistically injective) PRGs with polynomial stretch (mapping ℓ bits to ℓ_{PRG} bits for an appropriately chosen polynomial ℓ_{PRG}). Consider a log-depth circuit C that takes as input ℓ_{in}-bits and outputs ℓ-bits. To obfuscate circuit C with lock value $\alpha \in \{0,1\}^\ell$ and message msg, the GKW scheme first chooses PRG from the family and computes an "expanded" lock value $\beta = \mathrm{PRG}(\alpha)$. It then takes the circuit $\widehat{C} = \mathrm{PRG}(C(\cdot))$ that takes as input ℓ_{in}-bits and outputs ℓ_{PRG}-bits, and generates the permutation branching program representation of $\widehat{C}$. Let $\mathsf{BP}^{(i)}$ denote the branching program that computes i^{th} output bit of $\widehat{C}$. Since C and PRG are both log-depth circuits, we know (due to Barrington's theorem [Bar86]) that $\mathsf{BP}^{(i)}$ is of some polynomial length L and width 5.[4] The obfuscator continues by sampling $5\ell_{\mathrm{PRG}}$ matrices, for each level except the last one, using lattice trapdoor samplers such that all the matrices at any particular level share a common trapdoor. Let $\mathbf{B}_{j,k}^{(i)}$ denote the matrix corresponding to level j, state k of the i^{th} branching program $\mathsf{BP}^{(i)}$. Next, it chooses the top level matrices $\left\{\mathbf{B}_{L,1}^{(i)}, \ldots, \mathbf{B}_{L,5}^{(i)}\right\}$ for each $i \in [\ell_{\mathrm{PRG}}]$ uniformly at random subject to the following "sum-constraint":

$$\sum_{i:\ \beta_i=0} \mathbf{B}_{L,\mathsf{rej}^{(i)}}^{(i)} + \sum_{i:\ \beta_i=1} \mathbf{B}_{L,\mathsf{acc}^{(i)}}^{(i)} = \begin{cases} \mathbf{0}^{n\times m} & \text{if } \mathsf{msg} = 0, \\ \sqrt{q}\cdot\left[\mathbf{I}_n \,||\, \mathbf{0}^{n\times(m-n)}\right] & \text{if } \mathsf{msg} = 1. \end{cases}$$

Looking ahead, sampling the top level matrices in such a way helps to encode the expanded lock value β such that an evaluator can test for this relation if it has an input x such that $C(x) = \alpha$.

Next step in the obfuscation procedure is to encode the branching programs using the matrices and trapdoors sampled above. The idea is to choose a set of $\ell_{\mathrm{PRG}} \cdot L$ "transition matrices" $\{\mathbf{C}_j^{(i,0)}, \mathbf{C}_j^{(i,1)}\}_{i,j}$ such that each matrix $\mathbf{C}_j^{(i,b)}$ is short and can be used to evaluate its corresponding state transition permutation $\sigma_{j,b}^{(i)}$. The obfuscation of C is set to be the ℓ_{PRG} base-level matrices $\{\mathbf{B}_{0,1}^{(i)}\}_i$ and $\ell_{\mathrm{PRG}} \cdot L$ transition matrices $\{\mathbf{C}_j^{(i,0)}, \mathbf{C}_j^{(i,1)}\}_{i,j}$.

[3] Strictly speaking, [GKW17b, Appendix C] shows how to extend the message space for semi-statistically correct lockable obfuscation schemes. However, the same transformation also works for perfectly correct schemes.

[4] Recall, a permutation branching program of length L and width w can be represented using w states, $2L$ permutations $\sigma_{j,b}$ over states for each level $j \leq L$, an input-selector function $\mathsf{inp}(\cdot)$ which determines the input read at each level, and an accepting and rejecting state. The program execution starts at state $\mathsf{st} = 1$ of level 0, and iteratively carried out as $\mathsf{st} = \sigma_{i,b}(\mathsf{st})$ (where b is the input bit read at level i). Depending upon the final state (i.e., at level L), the program either accepts or rejects.

Evaluating the obfuscated program on input $x \in \{0,1\}^{\ell_{\rm in}}$ is analogous to evaluating the $\ell_{\rm PRG}$ branching programs on x. For each $i \in [\ell_{\rm PRG}]$, the evaluation algorithm first computes $\mathbf{M}_i = \mathbf{B}_{0,1}^{(i)} \cdot \prod_{j=1}^{L} \mathbf{C}_j^{(i, x_{\mathsf{inp}(j)})}$ and then sums them together as $\mathbf{M} = \sum_i \mathbf{M}_i$. To compute the final output, it looks at the entries of matrix $\mathbf{M}$, if all the entries are small (say less than $q^{1/4}$) it outputs 0, else if they are close to $\sqrt{q}$ it outputs 1, otherwise it outputs $\perp$.

To argue correctness, they first show that the matrix $\mathbf{M}$ computed by the evaluator is close to $\mathbf{\Gamma} \cdot \sum_i \mathbf{B}_{L, \mathsf{st}^{(i)}}^{(i)}$ where $\mathbf{\Gamma}$ is some low-norm matrix and $\mathsf{st}^{(i)}$ denotes the final state of $\mathsf{BP}^{(i)}$.[5] It is easy to verify that if $C(x) = \alpha$, then $\widehat{C}(x) = \beta$, and therefore

$$\mathbf{M} \approx \mathbf{\Gamma} \cdot \sum_i \mathbf{B}_{L, \mathsf{st}^{(i)}}^{(i)} = \begin{cases} \mathbf{0}^{n \times m} & \text{if } \mathsf{msg} = 0, \\ \sqrt{q} \cdot \left[\mathbf{\Gamma} \,||\, \mathbf{0}^{n \times (m-n)} \right] & \text{if } \mathsf{msg} = 1. \end{cases}$$

As a result, if $C(x) = \alpha$, then the evaluation is correct. However, it turns out that even when $C(x) \neq \alpha$ the evaluation algorithm could still output 0/1 (recall that if $C(x) \neq \alpha$, then the evaluation algorithm must output $\perp$). There are two sources of errors here.

Non-Injective PRGs. First, it is possible that the PRG chosen is not injective. In this event (which happens with negligible probability if PRG is chosen honestly), there exist two inputs $y \neq y'$ such that $\mathrm{PRG}(y) = \mathrm{PRG}(y')$. As a result, if there exist two inputs $x, x' \in \{0,1\}^{\ell_{\rm in}}$ such that $C(x) = y, C(x') = y'$, then the obfuscation of C with lock y and message msg, when evaluated on x', outputs msg instead of $\perp$. Note that this source of error can be eliminated if we use a perfectly injective PRG family instead of a statistically injective PRG family.

Sum-Constraints. The second source of error is due to the way we encode the lock value in the top-level matrices. Let $x \neq x'$ be two distinct inputs, and let $\alpha = C(x)$, $\alpha' = C(x')$, $\beta = \mathrm{PRG}(\alpha)$ and $\beta' = \mathrm{PRG}(\alpha')$. Suppose we obfuscate C with lock value α. Recall that the obfuscator samples the top-level matrices uniformly at random with the only constraint that the top-level matrices corresponding to the expanded lock value β either sum to 0 (if $\mathsf{msg} = 0$), else they sum to certain medium-ranged matrix (i.e., entries $\approx \sqrt{q}$). Now this corresponds to sampling all but one top-level matrix uniformly at random (and without any constraint), and that one special matrix such that the constraint is satisfied. Therefore, it is possible (although with small probability) that summing together the top-level matrices for string β' is close to top-level matrix sum for string β. That is,

$$\sum_{i:\ \beta_i = 0} \mathbf{B}_{L, \mathsf{rej}^{(i)}}^{(i)} + \sum_{i:\ \beta_i = 1} \mathbf{B}_{L, \mathsf{acc}^{(i)}}^{(i)} \approx \sum_{i:\ \beta'_i = 0} \mathbf{B}_{L, \mathsf{rej}^{(i)}}^{(i)} + \sum_{i:\ \beta'_i = 1} \mathbf{B}_{L, \mathsf{acc}^{(i)}}^{(i)}.$$

As a result, if we obfuscate C with lock α and message msg, and evaluate this on input x', then it could also output msg instead of $\perp$. This type of error is trickier to remove as it is crucial for security in the GKW construction that these matrices look completely random if one doesn't know the lock value α. To get around this issue, we provide an alternate top-level matrix sampling procedure that guarantees perfect correctness.

We next present our solutions to remove the above sources of imperfectness. First, we construct a perfectly injective PRG family that is secure under the LWE assumption. This resolves the first problem. Thereafter, we discuss our modifications to the GKW construction for resolving the *sum-constraint* error. Later we also briefly talk about our perfectly injective PRG family that is secure under the LPN assumption.

Perfectly injective PRG family. We will first show a perfectly injective PRG family based on the LWE assumption. The construction is a low-depth PRG family with unbounded (polynomial) stretch. The security of this construction relies on the Learning with Rounding (LWR) assumption, introduced by Banerjee et

[5] That is, $\mathsf{st}^{(i)} = \mathsf{acc}^{(i)}$ if $\widehat{C}(x)_i = 0$ and $\mathsf{rej}^{(i)}$ otherwise.

al. [BPR12], which in turn can be reduced to LWE (with subexponential modulus/error ratio). First, let us recall the LWR assumption. This assumption is associated with two moduli p, q where $p < q$. The modulus q is the modulus of computation, and p is the rounding modulus. Let $\lfloor \cdot \rceil_p$ denote a mapping from $\mathbb{Z}_q$ to $\mathbb{Z}_p$ which maps integers based on their higher order bits. The LWR assumption states that for a uniformly random secret vector $\mathbf{s} \in \mathbb{Z}_q^n$ and uniformly random matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, $\lfloor \mathbf{s}^T \cdot \mathbf{A} \rceil_p$ looks like a uniformly random vector in $\mathbb{Z}_p^m$, even when given $\mathbf{A}$. We will work with a 'binary secrets' version where the secret vector $\mathbf{s}$ is a binary vector.

Let us start by reviewing the PRG construction provided by Banerjee et al. [BPR12]. In their scheme, the setup algorithm first chooses two moduli $p < q$ and outputs a uniformly random $n \times m$ matrix $\mathbf{A}$ with elements from $\mathbb{Z}_q$. The PRG evaluation takes as input an n bit string $\mathbf{s}$ and outputs $\lfloor \mathbf{s}^T \cdot \mathbf{A} \rceil_p$, where $\lfloor x \rceil_p$ essentially outputs the higher order bits of x. Assuming m is sufficiently larger than n and moduli p, q are appropriately chosen, for a uniformly random matrix $\mathbf{A} \leftarrow \mathbb{Z}_q^{n \times m}$, the function $\lfloor \mathbf{s}^T \cdot \mathbf{A} \rceil_p$ is injective with high probability (over the choice of $\mathbf{A}$). In order to achieve perfect injectivity, we sample the public matrix $\mathbf{A}$ in a special way.

In our scheme, the setup algorithm chooses a uniformly random matrix $\mathbf{B}$ and a low norm matrix $\mathbf{C}$. It sets $\mathbf{D}$ to be a diagonal matrix with medium-value entries ($\mathbf{D}$ is a fixed deterministic matrix). It sets $\mathbf{A} = [\mathbf{B} \mid \mathbf{B} \cdot \mathbf{C} + \mathbf{D}]$ and outputs it as part of the public parameters, together with the LWR moduli p, q. To evaluate the PRG on input $\mathbf{s} \in \{0,1\}^n$, one outputs $\mathbf{y} = \lfloor \mathbf{s}^T \cdot \mathbf{A} \rceil_p$. Intuitively, the $\mathbf{D}$ matrix acts as a error correcting code, and if $\mathbf{s}_1 \neq \mathbf{s}_2$, then there is at least one coordinate such that $\lfloor \mathbf{s}_1^T \cdot \mathbf{D} \rceil_p$ and $\lfloor \mathbf{s}_2^T \cdot \mathbf{D} \rceil_p$ are far apart.

Suppose $\mathbf{s}_1$ and $\mathbf{s}_2$ are two bitstrings such that $\lfloor \mathbf{s}_1^T \cdot \mathbf{A} \rceil_p = \lfloor \mathbf{s}_2^T \cdot \mathbf{A} \rceil_p$. Then $\lfloor \mathbf{s}_1^T \cdot \mathbf{B} \rceil_p = \lfloor \mathbf{s}_2^T \cdot \mathbf{B} \rceil_p$, and as a result, $\lfloor \mathbf{s}_1^T \cdot \mathbf{B} \cdot \mathbf{C} \rceil_p$ and $\lfloor \mathbf{s}_2^T \cdot \mathbf{B} \cdot \mathbf{C} \rceil_p$ have close enough entries as $\mathbf{C}$ has small entries. However, this implies that $\lfloor \mathbf{s}_1^T \cdot \mathbf{D} \rceil_p$ and $\lfloor \mathbf{s}_2^T \cdot \mathbf{D} \rceil_p$ also have close enough entries, which implies that $\mathbf{s}_1 = \mathbf{s}_2$.

Pseudorandomness follows from the observation that $\mathbf{A}$ looks like a uniformly random matrix. Once we replace $[\mathbf{B} \mid \mathbf{B} \cdot \mathbf{C} + \mathbf{D}]$ with a uniformly random matrix $\mathbf{A}$, we can use the binary secrets version of LWR to argue that $\mathbf{s}^T \cdot \mathbf{A}$ is indistinguishable from a uniformly random vector. This is discussed in detail in Section 3.

Relation to the perfectly binding commitment scheme of [GHKW17]: The perfectly injective PRG family outlined above builds upon some core ideas from the perfectly binding commitments schemes in [GHKW17]. Below, we will describe the constructions from [GHKW17], and discuss the main differences in our PRG schemes.

In the LWE based commitment scheme, the sender first chooses a modulus q, matrices $\mathbf{B}, \mathbf{C}, \mathbf{D}$ and $\mathbf{E}$ of dimensions $n \times n$, where $\mathbf{B}$ is a uniformly random matrix, entries in $\mathbf{C}$, $\mathbf{E}$ are drawn from the low norm noise distribution, and $\mathbf{D}$ is some fixed diagonal matrix with medium-value entries. It sets $\mathbf{A} = [\mathbf{B} \,||\, \mathbf{B} \cdot \mathbf{C} + \mathbf{D} + \mathbf{E}]$. Next, it chooses a vector $\mathbf{s}$ from the noise distribution, vector $\mathbf{w}$ uniformly at random, vector $\mathbf{e}$ from the noise distribution and f from the noise distribution. To commit to a bit b, it sets $\mathbf{y} = \mathbf{A}^T \cdot \mathbf{s} + \mathbf{e}$, $z = \mathbf{w}^T \cdot \mathbf{s} + f + b(q/2)$, and the commitment is $(\mathbf{A}, \mathbf{w}, \mathbf{y}, z)$. The opening simply consists of the randomness used for constructing the commitment.

The main differences between our PRG construction and their commitment scheme are as follows: (i) we need to separate out their initial commitment step into PRG setup and evaluation phase, (ii) since the PRG evaluation is deterministic, we cannot add noise (unlike in the case of commitments). Therefore, we need to use Learning with Rounding. Finally, we need to carefully choose the rounding modulus p as we want to ensure that the rounding operation does not round off the contribution from the special matrix $\mathbf{D}$ while it still allowing us to reduce to the LWR assumption.

Sum-constraint on the top-level matrices. We will now discuss how the top-level matrices can be sampled to ensure perfect correctness. In order to do so, let us first consider the following simplified problem which captures the essence of the issue. Given a string $\beta \in \{0,1\}^\ell$, we wish to sample 2ℓ matrices $\{\mathbf{M}_{i,b}\}_{i \in [\ell], b \in \{0,1\}}$ such that they satisfy the following three constraints:

1. $\sum_i \mathbf{M}_{i,\beta_i}$ has 'small' entries (say $< q^{1/4}$).

2. For **all** $\beta' \neq \beta$, $\sum_i \mathbf{M}_{i,\beta'_i}$ has 'large' entries (say greater than $q^{1/2}$).
3. For a uniformly random choice of string β, the set of 2ℓ matrices $\{\mathbf{M}_{i,b}\}_{i,b}$ 'look' like random matrices.

In the GKW construction, the authors use a simple sampler that the sampled matrices satisfy the first constraint, and by applying the Leftover Hash Lemma (LHL) they also show that the corresponding matrices satisfy the third constraint. However, to achieve perfect correctness, we need to build a matrix sampler such that its output always satisfy all the three constraints. To this end, we show that by carefully embedding LWE samples inside the output matrices we can achieve the second constraint as well. We discuss our approach in detail below.

We now define a sampler Samp that takes an ℓ-bit string β as input, and outputs 2ℓ matrices satisfying all the above constraints, assuming the Learning with Errors assumption (in addition to relying on LHL). The sampler first chooses 2ℓ uniformly random *square* matrices $\{\mathbf{A}_{i,b}\}_{i\in[\ell],b\in\{0,1\}}$ subject to the constraint that $\sum_i \mathbf{A}_{i,\beta_i} = \mathbf{0}^{n\times n}$. This can be achieved by simply sampling $2\ell - 1$ uniformly random $n \times n$ matrices, and setting $\mathbf{A}_{\ell,\beta_\ell} = -\sum_{i<\ell} \mathbf{A}_{i,\beta_i}$. Let $\mathbf{D} = q^{3/4}\left[\mathbf{I}_n \,||\, \mathbf{0}^{n\times(m-2n)}\right]$ be a $n \times (m-n)$ matrix with a few 'large' entries. The sampler then chooses a low norm $n \times (m-n)$ matrix $\mathbf{S}$ and low-norm $n \times (m-n)$ error matrices $\{\mathbf{E}_{i,b}\}_{i\in[\ell],b\in\{0,1\}}$. It sets the 2ℓ output matrices as

$$\mathbf{M}_{i,b} = \begin{cases} [\mathbf{A}_{i,b} \,||\, \mathbf{A}_{i,b}\cdot\mathbf{S} + \mathbf{E}_{i,b}] & \text{if } b = \beta_i \\ [\mathbf{A}_{i,b} \,||\, \mathbf{A}_{i,b}\cdot\mathbf{S} + \mathbf{E}_{i,b} + \mathbf{D}] & \text{if } b = 1-\beta_i \end{cases}$$

In short, our sampler samples the first n columns of the output matrix in a similar way to GKW scheme, whereas the remaining $(m-n)$ columns are sampled in a special way such that if we sum up the matrices corresponding to string β then the last $(m-n)$ columns of the summed matrix have small entries, wheras summing up matrices corresponding to any other string $\beta' \neq \beta$, the last $(m-n)$ columns of the summed matrix have distinguishably large entries. Below we briefly argue why our sampler satisfies the three properties specified initially.

1. (First property): Note that $\sum_i \mathbf{A}_{i,\beta_i} = \mathbf{0}^{n\times n}$, therefore we have that

$$\mathbf{M}_\beta = \sum_i \mathbf{M}_{i,\beta_i} = \left[\mathbf{0}^{n\times n} \,||\, \mathbf{0}^{n\times n}\cdot\mathbf{S} + \sum_i \mathbf{E}_{i,\beta_i}\right] = \left[\mathbf{0}^{n\times n} \,||\, \sum_i \mathbf{E}_{i,\beta_i}\right].$$

Since the error matrices are drawn from a low-norm distribution, the entries of $\mathbf{M}_\beta$ are 'small'.

2. (Second property): We need to check that $\mathbf{M}_{\beta'} = \sum_i \mathbf{M}_{i,\beta'_i}$ has 'large' entries for $\beta' \neq \beta$. Suppose β and β' differ at t positions ($t > 0$). Then

$$\sum_i \mathbf{M}_{i,\beta'_i} = \left[\sum_i \mathbf{A}_{\beta'} \,||\, \mathbf{A}_{\beta'}\cdot\mathbf{S} + \mathbf{E}_{\beta'} + t\cdot\mathbf{D}\right],$$

where $\mathbf{A}_{\beta'} = \sum_i \mathbf{A}_{i,\beta'_i}$ and $\mathbf{E}_{\beta'} = \sum_i \mathbf{E}_{i,\beta'_i}$. If $\mathbf{A}_{\beta'}$ has large entries (greater than $q^{1/2}$), then we are done. On the other hand, if $\mathbf{A}_{\beta'}$ has small entries (less than $q^{1/2}$), then we can argue that $\mathbf{A}_{\beta'}\cdot\mathbf{S} + \mathbf{E}_{\beta'}$ also has entries less than $q^{3/4}$, and therefore $\mathbf{A}_{\beta'}\cdot\mathbf{S} + \mathbf{E}_{\beta'} + t\cdot\mathbf{D}$ has large entries. This implies that $\mathbf{M}_{\beta'}$ has large entries, and hence the second constraint is also satisfied.

3. (Third property): To argue about the third property, we use the LWE assumption in conjunction with LHL. First, we can argue that the $\{\mathbf{A}_{i,b}\}$ matrices look like uniformly random matrices (using the leftover hash lemma). Next, using the LWE assumption, we can show that $\{[\mathbf{A}_{i,b} \,||\, \mathbf{A}_{i,b}\cdot\mathbf{S} + \mathbf{E}_{i,b}]\}_{i,b}$ are indistinguishable from 2ℓ uniformly random matrices, and hence the third property is also satisfied.

We can also modify the above sampler slightly such that $\sum_i \mathbf{M}_{i,\beta_i}$ has 'medium' entries (that is, entries within the range $[q^{1/4}, q^{1/2})$). The sampler chooses random matrices $\{\mathbf{A}_{i,b}\}_{i,b}$ subject to the constraint that $\sum_i \mathbf{A}_{i,\beta_i} = q^{1/4}\mathbf{I}_n$, and the remaining steps are same as above. Let $\mathsf{Samp}_{\mathrm{med}}$ be the sampler for this 'medium-entries' variant.

We observe that if we plug in these samplers into the GKW scheme for sampling their top-level matrices, then that leads to a perfectly correct lockable obfuscation scheme. Specifically, let α be the lock used, PRG chosen from a perfectly injective PRG family, and $\beta = \mathrm{PRG}(\alpha)$ be the expanded lock value. The obfuscation scheme chooses matrices $\{\mathbf{M}_{i,b}\}_{i,b}$ using either Samp or $\mathsf{Samp}_{\mathrm{med}}$ depending on the message msg. That is, if $\mathsf{msg} = 0$, it chooses $\{\mathbf{M}_{i,b}\}_{i,b} \leftarrow \mathsf{Samp}(\beta)$, else it chooses $\{\mathbf{M}_{i,b}\}_{i,b} \leftarrow \mathsf{Samp}_{\mathrm{med}}(\beta)$. It then sets $\mathbf{B}^{(i)}_{L,\mathsf{acc}^{(i)}} = \mathbf{M}_{i,1}$ and $\mathbf{B}^{(i)}_{L,\mathsf{rej}^{(i)}} = \mathbf{M}_{i,0}$ for each $i \in [\ell_{\mathrm{PRG}}]$. From the properties of $\mathsf{Samp}/\mathsf{Samp}_{\mathrm{med}}$, it follows that

$$\mathbf{M}_\beta = \sum_i \mathbf{M}_{i,\beta_i} = \sum_{i:\ \beta_i=0} \mathbf{B}^{(i)}_{L,\mathsf{rej}^{(i)}} + \sum_{i:\ \beta_i=1} \mathbf{B}^{(i)}_{L,\mathsf{acc}^{(i)}},$$

which has 'low' or 'medium' norm depending on msg bit. The remaining top level matrices are chosen uniformly at random. Everything else stays the same as in the GKW scheme.

For completeness, we now check that this scheme indeed satisfies perfect correctness. Consider an obfuscation of circuit C with lock α and message msg. If this obfuscation is evaluated on input x such that $C(x) = \alpha$, then the evaluation outputs msg as expected. If $C(x) = \alpha' \neq \alpha$, then $\mathrm{PRG}(C(x)) = \beta' \neq \beta$ (since the PRG is injective). This means the top level sum is

$$\sum_{i:\ \beta'_i=0} \mathbf{B}^{(i)}_{L,\mathsf{rej}^{(i)}} + \sum_{i:\ \beta'_i=1} \mathbf{B}^{(i)}_{L,\mathsf{acc}^{(i)}} = \sum_i \mathbf{M}_{i,\beta'_i},$$

Using the second property of $\mathsf{Samp}/\mathsf{Samp}_{\mathrm{med}}$, we know that this sum has 'large' entries, and therefore the evaluation outputs $\perp$. This completes our perfect correctness argument. Now for proving that our modification still give a secure lockable obfuscation, we do not re-derive a completely new security proof but instead we show that no PPT attacker can distinguish an obfuscated program generated using our scheme from the one generated by using the GKW scheme. Now combining this claim with the fact that the GKW scheme is secure under LWE assumption, we get that our scheme is also secure. Very briefly, the idea behind indistinguishability of these two schemes is that since the lock α is chosen uniformly at random, then $\mathrm{PRG}(\alpha)$ is computationally indistinguishable from a uniformly random string β, and thus these top level matrices also look like uniformly random matrices for uniformly random β (using the third property of $\mathsf{Samp}/\mathsf{Samp}_{\mathrm{med}}$). Now to complete argument we show the same hold for GKW scheme as well, thereby completing the proof. More details on this are provided in the main body.

Perfectly Injective PRGs from the LPN assumption. Finally, we also build a family of perfectly injective PRGs based on the Learning Parity with Noise assumption. While the focus of this work has been getting an end-to-end LWE solution for perfectly correct lockable obfuscation, we also build perfectly injective PRGs based on the LPN assumption, which could be of independent interest. Recently, there has been a surge of interest towards new constructions of cryptographic primitives based on LPN [YS16, YZ16, YZW+17, DGHM18, BLSV18, BLVW18], and we feel that our perfectly injective PRGs fit this theme. Our LPN solution uses a low-noise variant ($\beta \approx \frac{1}{\sqrt{n}}$) of the LPN assumption that has been used in previous public key encryption schemes [Ale03]. Below we briefly sketch the main ideas behind our PRG construction.

To build perfectly injective PRGs from LPN, we take a similar approach to one taken in the LWE case. The starting idea is to use the PRG seed (as before) as the secret vector $\mathbf{s}$ and compute the PRG evaluation as $\mathbf{B}^T\mathbf{s}$) but now, unlike the LWE case, we do not have any rounding equivalent for LPN, that is we do not know how to avoid generating the error vector $\mathbf{e}$ during PRG evaluation. Therefore, to execute the idea we provide an (efficient) *injective* sampler for error vectors which takes as input a bit string and outputs an error vector $\mathbf{e}$ of appropriate dimension. (The injectivity property here states that the mapping between bit strings and the error vectors is injective.) So now in our PRG evaluation the input string is first divided in two disjoint components where the first component is directly interpreted as the secret vector $\mathbf{s}$ and second component is used to sample the error vector $\mathbf{e}$ using our injective sampler.

Although at first it might seem that building an injective sampler might not be hard, however it turns out there are a couple of subtle issues that we have take care of while proving security as well as perfect injectivity. Concretely, for self-composability of our PRG (i.e., building PRGs which take as input bit strings of fixed length instead having a special domain sampling algorithm), we require that the size of support of distribution of error vectors $\mathbf{e}$ used is a 'perfect power of two'. As otherwise we can not hope to build a perfectly injective (error vector) sampler which takes as input a fixed length bit string and outputs the corresponding error vector. Now we know that the size of support of noise distribution in the LPN assumption might not be a perfect power of two, thus we might not be able to injectively sample error vectors from the fixed length bit strings. To resolve this issue, we define an alternate assumption which we call the 'restricted-exact-LPN' assumption and show that (a) it is as hard as standard LPN, (b) sufficient for our proof to go through, and (c) has an efficiently enumerable noise distribution whose support size is a perfect power of two (i.e., we can define an efficient injective error sampler for its noise distribution). More details are provided later in Section 5.

1.2 Related Works on Perfect Correctness

In this section, we discuss some related work and approaches for achieving perfect correctness for lockable obfuscation and its applications.

Perfect Correctness via Derandomization. Bitansky and Vaikuntanathan [BV17] showed how to transform any obfuscation scheme (and a large class of cryptosystems) to remove correctness errors using Nisan-Wigderson (NW) PRGs [NW94]. In their scheme, the obfuscator runs the erroneous obfuscation algorithm sufficiently many times, and for each execution of the obfuscator, the randomness used is derived pseudorandomly (by adding the randomness derived from the NW PRGs and the randomness from a standard cryptographic PRG). As the authors show, such a transformation leads to a perfectly correct scheme as long as certain circuit lower bound assumptions hold (in particular, they require that the NW-PRGs can fool certain bounded-size circuits). Our solution, on the other hand, does not rely on additional assumptions as well as it is as efficient as existing (imperfect) lockable obfuscation constructions [GKW17a, WZ17].

Using a Random Oracle for generating randomness. A heuristic approach to prevent the obfuscator from using malicious randomness is to generate the random coins using a hash function H applied on the circuit. Such a heuristic might suffice for some applications such as the public auditing example discussed previously, but it does not seem to provide provable security in others. Note that our construction with perfect correctness is proven secure in the standard model, and does not need rely on ROs or a CRS.

Lastly, we want to point out that in an earlier work by Brakerski and Vaikuntanathan [BV16] it was shown how to transform any obfuscation scheme that has statistical correctness on $(1/2+\epsilon)$ fraction of inputs (for some non-negligible ϵ) into a scheme that has statistical correctness for all inputs. However, this does not achieve perfect correctness. It is an interesting question whether their approach could be extended to achieve perfect correctness.

2 Preliminaries

In this section, we will introduce some notations and preliminaries required for our work.

2.1 Notations

We will be using bold lowercase vectors to denote vectors and bold uppercase vectors for matrices. For any set $\mathcal{S}$, $s \leftarrow \mathcal{S}$ denotes a uniformly random element drawn from $\mathcal{S}$. Similarly, for any distribution $\mathcal{D}$, $x \leftarrow \mathcal{D}$ denotes an element drawn from distribution $\mathcal{D}$.

For any modulus $p > 2$, let $\mathbb{Z}_p$ denote the set $\{-\lfloor p/2 \rfloor, -\lfloor p/2 \rfloor + 1, \ldots, \lfloor p/2 \rfloor - 1\}$, and for any integer x, $x \mod p$ maps x to $\mathbb{Z}_p$. For any real number $x \in \mathbb{R}$, let $\lfloor x \rceil$ denote the integer closest to x. For any vector $\mathbf{v} \in \mathbb{Z}_2^n$, we use $\mathsf{int}(\mathbf{v})$ to denote its integer representation, i.e. $\mathsf{int}(\mathbf{v}) = \sum_{i=1}^{n} v_i 2^{i-1}$ where v_i denotes the i^{th} element of $\mathbf{v}$. Similarly, for bit strings $s \in \{0,1\}^n$, we use $\mathsf{int}(s)$ to denote its integer representation.

Min-Entropy and Randomness Extraction. The min-entropy of a random variable X is defined as $\mathbf{H}_\infty(X) \overset{\text{def}}{=} -\log_2(\max_x \Pr[X = x])$. Let $\mathsf{SD}(X, Y)$ denote the statistical distance between two random variables X and Y. Below we state the Leftover Hash Lemma (LHL) from [HILL99, DRS04, DORS08].

Theorem 2.1. Let $\mathcal{H} = \{h : X \to Y\}_{h \in \mathcal{H}}$ be a universal hash family, then for any random variable W taking values in X, the following holds

$$\mathsf{SD}\left((h, h(W)), (h, U_Y)\right) \leq \frac{1}{2}\sqrt{2^{-\mathbf{H}_\infty(W)} \cdot |Y|}\ .$$

We will use the following corollary, which follows from the Leftover Hash Lemma.

Corollary 2.1. Let $\ell > m \cdot n \log_2 q + \omega(\log n)$ and q a prime. Let $\mathbf{R}$ be an $k \times m$ matrix chosen as per distribution $\mathcal{R}$, where $k = k(n)$ is polynomial in n and $\mathbf{H}_\infty(\mathcal{R}) = \ell$. Let $\mathbf{A}$ and $\mathbf{B}$ be matrices chosen uniformly in $\mathbb{Z}_q^{n \times k}$ and $\mathbb{Z}_q^{n \times m}$, respectively. Then the statistical distance between the following distributions is negligible in n.

$$\{(\mathbf{A}, \mathbf{A} \cdot \mathbf{R})\} \approx_s \{(\mathbf{A}, \mathbf{B})\}$$

Lattices. An m-dimensional lattice $\mathcal{L}$ is a discrete additive subgroup of $\mathbb{R}^m$. Given positive integers n, m, q and a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, we let $\Lambda_q^\perp(\mathbf{A})$ denote the lattice $\{\mathbf{x} \in \mathbb{Z}^m : \mathbf{A} \cdot \mathbf{x} = \mathbf{0} \mod q\}$. For $\mathbf{u} \in \mathbb{Z}_q^n$, we let $\Lambda_q^{\mathbf{u}}(\mathbf{A})$ denote the coset $\{\mathbf{x} \in \mathbb{Z}^m : \mathbf{A} \cdot \mathbf{x} = \mathbf{u} \mod q\}$.

Discrete Gaussians. Let σ be any positive real number. The Gaussian distribution $\mathcal{D}_\sigma$ with parameter σ is defined by the probability distribution function $\rho_\sigma(\mathbf{x}) = \exp(-\pi \left\|\mathbf{x}\right\|^2 / \sigma^2)$. For any set $\mathcal{L} \subset \mathcal{R}^m$, define $\rho_\sigma(\mathcal{L}) = \sum_{\mathbf{x} \in \mathcal{L}} \rho_\sigma(\mathbf{x})$. The discrete Gaussian distribution $\mathcal{D}_{\mathcal{L},\sigma}$ over $\mathcal{L}$ with parameter σ is defined by the probability distribution function $\rho_{\mathcal{L},\sigma}(\mathbf{x}) = \rho_\sigma(\mathbf{x}) / \rho_\sigma(\mathcal{L})$ for all $\mathbf{x} \in \mathcal{L}$.

The following lemma (Lemma 4.4 of [MR07], [GPV08]) shows that if the parameter σ of a discrete Gaussian distribution is small, then any vector drawn from this distribution will be short (with high probability).

Lemma 2.1. Let m, n, q be positive integers with $m > n$, $q \geq 2$. Let $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ be a matrix of dimensions $n \times m$, $\sigma = \tilde{\Omega}(n)$ and $\mathcal{L} = \Lambda_q^\perp(\mathbf{A})$. Then

$$\Pr[\|\mathbf{x}\| > \sqrt{m} \cdot \sigma : \mathbf{x} \leftarrow \mathcal{D}_{\mathcal{L},\sigma}] \leq \mathrm{negl}(n).$$

Truncated Discrete Gaussians. The truncated discrete Gaussian distribution over $\mathbb{Z}^m$ with parameter σ, denoted by $\widetilde{\mathcal{D}}_{\mathbb{Z}^m,\sigma}$, is same as the discrete Gaussian distribution $\mathcal{D}_\sigma$ except it outputs $\mathbf{0}$ vector whenever the ℓ_∞ norm exceeds $\sqrt{m} \cdot \sigma$. Note that, by definition, $\widetilde{\mathcal{D}}_{\mathbb{Z}^m,\sigma}$ is $\sqrt{m} \cdot \sigma$-bounded. Also, note that $\widetilde{\mathcal{D}}_{\mathbb{Z}^m,\sigma} \approx_s \mathcal{D}_{\mathbb{Z}^m,\sigma}$.

2.2 Learning with Errors Assumption

The Learning with Errors (LWE) problem was introduced by Regev [Reg05]. The LWE problem has four parameters: the dimension of the lattice n, the number of samples m, the modulus q and the error distribution $\chi = \chi(n)$.

Let n, m and q be positive integers and χ a noise distribution over $\mathbb{Z}_q$. The Learning with Errors assumption (n, m, q, χ)-LWE, parameterized by n, m, q, χ, states that the following distributions are computationally indistinguishable:

$$\left\{ (\mathbf{A}, \mathbf{s}^T \cdot \mathbf{A} + \mathbf{e}^T) \ : \ \begin{array}{l} \mathbf{A} \leftarrow \mathbb{Z}_q^{n \times m}, \\ \mathbf{s} \leftarrow \mathbb{Z}_q^n, \mathbf{e} \leftarrow \chi^m \end{array} \right\} \approx_c \left\{ (\mathbf{A}, \mathbf{u}^T) \ : \ \begin{array}{l} \mathbf{A} \leftarrow \mathbb{Z}_q^{n \times m}, \\ \mathbf{u} \leftarrow \mathbb{Z}_q^m \end{array} \right\}$$

Under a quantum reduction, Regev [Reg05] showed that for certain noise distributions, LWE is as hard as worst case lattice problems such as the decisional approximate shortest vector problem (GapSVP) and approximate shortest independent vectors problem (SIVP). Later works [Pei09, BLP$^+$13a] showed classical reductions from LWE to GapSVP_γ.

These works show that for B-bounded discretized Gaussian error distributions χ, solving (n, m, q, χ)-LWE is as hard as approximating GapSVP_γ and SIVP_γ to a factor of $\tilde{O}(n \cdot q/B)$. Given the current state of art in lattice algorithms, GapSVP_γ and SIVP_γ are believed to be hard for $\gamma = \tilde{O}(2^{n^\epsilon})$ (for fixed $\epsilon \in (0, 1/2)$), and therefore (n, m, q, χ)-LWE is believed to be hard for B-bounded discretized Gaussian error distributions χ with $B = 2^{-n^\epsilon} \cdot q \cdot \mathsf{poly}(n)$.

LWE with Short Secrets. In this work, we will be using a variant of the LWE problem called *LWE with Short Secrets*. In this variant, introduced by Applebaum et al. [ACPS09], the secret vector is also chosen from the noise distribution χ. They showed that this variant is as hard as LWE for sufficiently large number of samples m.

Assumption 1 (LWE with Short Secrets)**.** Let n, m, k and q be positive integers and χ a noise distribution on $\mathbb{Z}$. The LWE with Short Secrets assumption (n, m, k, q, χ)-$\mathsf{LWE\text{-}ss}$, parameterized by n, m, k, q, χ, states that the following distributions are computationally indistinguishable [6]:

$$\left\{ (\mathbf{A}, \mathbf{A} \cdot \mathbf{S} + \mathbf{E}) \ : \ \begin{array}{c} \mathbf{A} \leftarrow \mathbb{Z}_q^{m \times n}, \\ \mathbf{S} \leftarrow \chi^{n \times k}, \mathbf{E} \leftarrow \chi^{m \times k} \end{array} \right\} \approx_c \left\{ (\mathbf{A}, \mathbf{U}) \ : \ \begin{array}{c} \mathbf{A} \leftarrow \mathbb{Z}_q^{m \times n}, \\ \mathbf{U} \leftarrow \mathbb{Z}_q^{m \times k} \end{array} \right\}.$$

2.3 The Learning with Rounding (LWR) Assumption

Let $2 \leq p \leq q$ be two moduli. For any integer x in $\mathbb{Z}_q$, let $\lfloor x \rceil_p$ denote $\lfloor (p/q) \cdot x \rceil$. This notion can analogously be extended to vectors; that is, for any vector $\mathbf{y} \in \mathbb{Z}_q^n$, let $\mathbf{w} = \lfloor \mathbf{y} \rceil_p$ denote the vector in $\mathbb{Z}_p^n$ where $\mathbf{w}_j = \lfloor \mathbf{y}_j \rceil_p$ for all $j \in \{1, 2, \ldots, n\}$.

Assumption 2 (LWR)**.** The Learning with Rounding assumption with moduli q, p and dimension n states that the following distributions are computationally indistinguishable:

$$\left\{ \left(\mathbf{A}, \lfloor \mathbf{s}^T \cdot \mathbf{A} \rceil_p \right) \ : \ \mathbf{A} \leftarrow \mathbb{Z}_q^{n \times m}, \mathbf{s} \leftarrow \mathbb{Z}_q^n \right\} \approx_c \left\{ (\mathbf{A}, \mathbf{u}) \ : \ \mathbf{A} \leftarrow \mathbb{Z}_q^{n \times m}, \mathbf{u} \leftarrow \mathbb{Z}_p^m \right\}$$

The LWR assumption was first introduced by Banerjee, Peikert and Rosen [BPR12] (BPR). They showed that for certain settings of the moduli p, q, the LWR problem is as hard as LWE with subexponential modulus.

Theorem 2.2 ([BPR12])**.** Let $2 \leq p \leq q$ be two moduli and n the dimension such that q/p is superpolynomial in n. Then, assuming the LWE problem is hard for modulus q, dimension n and discrete Gaussian error distribution with parameter $\sigma = \mathsf{poly}(n)$, the LWR problem is hard for moduli p, q and dimension n.

We would like to point out that later works [AKPW13, BGM$^+$16] gave tighter reductions which enabled a larger range of parameters, specifically they allowed a polynomial modulus and modulus-to-error ratio. However the choice of modulus q must linearly scale with the number of samples m. In our PRG construction, the number of samples is known at setup time, therefore we could also use a polynomial modulus in our PRG construction. For simplicity of exposition, we only consider parameters provided by the BPR reduction.

In this work, we will be considering an LWR variant where the secret vector is a uniformly random binary vector. Using the BPR reduction, we can show that this problem is as hard as LWE with secret vector drawn from uniform distribution on binary vectors. Finally, using the reductions from [BLP$^+$13b, MP12], we can show that the binary-LWE problem is as hard as standard LWE (on lower dimension).

[6] Applebaum et al. showed that $\{(\mathbf{A}, \mathbf{s}^T \cdot \mathbf{A} + \mathbf{e}) : \mathbf{A} \leftarrow \mathbb{Z}_q^{n \times m}, \mathbf{s} \leftarrow \chi^n, \mathbf{e} \leftarrow \chi^m\} \approx_c \{(\mathbf{A}, \mathbf{u}) : \mathbf{A} \leftarrow \mathbb{Z}_q^{n \times m}, \mathbf{u} \leftarrow \mathbb{Z}_q^m\}$, assuming LWE is hard. However, by a simple hybrid argument, we can replace vectors $\mathbf{s}, \mathbf{e}, \mathbf{u}$ with matrices $\mathbf{S}, \mathbf{E}, \mathbf{U}$ of appropriate dimensions.

Assumption 3 (Binary LWR)**.** The Binary Learning with Rounding assumption with moduli q, p and dimension n states that the following distributions are computationally indistinguishable:

$$\left\{\left(\mathbf{A}, \lfloor \mathbf{s}^T \cdot \mathbf{A} \rceil_p\right) \; : \; \mathbf{A} \leftarrow \mathbb{Z}_q^{n \times m}, \mathbf{s} \leftarrow \mathbb{Z}_2^n\right\} \approx_c \left\{(\mathbf{A}, \mathbf{u}) \; : \; \mathbf{A} \leftarrow \mathbb{Z}_q^{n \times m}, \mathbf{u} \leftarrow \mathbb{Z}_p^m\right\}$$

Theorem 2.3 ([BPR12, BLP+13b, MP12])**.** Let $2 \leq p \leq q$ be two moduli and n the dimension such that q/p is superpolynomial in n. Then, assuming the LWE problem is hard for modulus q, dimension $n/\log q$ and discrete Gaussian error distribution with parameter $\sigma = \mathsf{poly}(n)$, the Binary LWR problem is hard for moduli p, q and dimension n.

As mentioned before, we can also choose parameters such that q and p are polynomials by relying on [AKPW13, BGM+16].

2.4 The Learning Parity with Noise (LPN) Assumption

The learning parity with noise is the binary ($\mathbb{Z}_2$) equivalent of the LWE problem. The search version of this problem requires one to solve a set of random linear equations perturbed by noise, and a decision version can be defined as in LWE. This problem is parameterized by the dimension n, the number of samples m and the error distribution. Each component of the error vector is chosen independently from the Bernoulli distribution with parameter p for $0 < p < 1/2$. Clearly, if $p = 1/2$, then the LPN distribution is identical to the uniform distribution. If $p = O(1/n)$, then an adversary can distinguish between the LPN distribution and the uniform distribution, given sufficiently many samples. Intuitively, the decision problem gets easier as p decreases.

Assumption 4 (Learning Parity with Noise)**.** Let n, m be positive integers and p be a real number such that $p < 1/2$. The (Decision) Learning Parity with Noise assumption $\mathsf{LPN}_{n,m,p}$, parameterized by the dimension of secret vector n, number of samples m and the error probability p, states that the following distributions are computationally indistinguishable:

$$\left\{(\mathbf{A}, \mathbf{A}^T\mathbf{s} + \mathbf{e}) \; : \; \begin{array}{l} \mathbf{A} \leftarrow \mathbb{Z}_2^{n \times m}, \\ \mathbf{s} \leftarrow \mathbb{Z}_2^n, \mathbf{e} \leftarrow \mathsf{Ber}_p^m \end{array}\right\} \approx_c \left\{(\mathbf{A}, \mathbf{u}) \; : \; \begin{array}{l} \mathbf{A} \leftarrow \mathbb{Z}_2^{n \times m}, \\ \mathbf{u} \leftarrow \mathbb{Z}_2^m \end{array}\right\}$$

Knapsack-LPN. In this work, we will be using a variant of LPN called Knapsack-LPN. For certain range of parameters, this variant can be shown to be equivalent to LPN [MM11].

Assumption 5 (Knapsack Learning Parity with Noise)**.** Let n, m be positive integers and p be a real number such that $p < 1/2$. The Knapsack Learning Parity with Noise assumption $\mathsf{KLPN}_{n,m,p}$, parameterized by integers n, m and p, states that the following distributions are computationally indistinguishable:

$$\left\{(\mathbf{A}, \mathbf{AE}) \; : \; \begin{array}{l} \mathbf{A} \leftarrow \mathbb{Z}_2^{n \times m}, \\ \mathbf{E} \leftarrow \mathsf{Ber}_p^{m \times m} \end{array}\right\} \approx_c \left\{(\mathbf{A}, \mathbf{B}) \; : \; \begin{array}{l} \mathbf{A} \leftarrow \mathbb{Z}_2^{n \times m}, \\ \mathbf{B} \leftarrow \mathbb{Z}_2^{n \times m} \end{array}\right\}$$

Clearly, if $n > m$, then the KLPN problem is easy. However, if $m > 2n$, then the KLPN problem is as hard as the LPN problem. In particular, there exists a reduction from $\mathsf{LPN}_{n,m,p}$ to $\mathsf{KLPN}_{m-n,m,p}$ as shown by [MM11].

Exact-LPN. Jain et al. [JKPT12] defined another variant of LPN which they called exact-LPN (or xLPN). The $\mathsf{xLPN}_{n,m,p}$ problem is defined exactly like the $\mathsf{LPN}_{n,m,p}$ problem, except the error vector is drawn uniformly from the distribution of vectors with hamming weight *exactly* $\lfloor mp \rceil$ (i.e., not just in expectation). Formally, the decision version of $\mathsf{xLPN}_{n,m,p}$ can be stated as follows where $\chi_{m,p}^{(\mathsf{e})} = \{\mathbf{v} \in \mathbb{Z}_2^m : \mathsf{HW}(\mathbf{v}) = \lfloor mp \rceil\}$ (i.e., the set of length m vectors with hamming weight $\lfloor mp \rceil$).

Assumption 6 (Exact Learning Parity with Noise)**.** Let n, m be positive integers and p be a real number such that $p < 1/2$. The (Decision) Exact Learning Parity with Noise assumption $\mathsf{LPN}_{n,m,p}$, parameterized by the dimension of secret vector n, number of samples m and the error probability p, states that the following distributions are computationally indistinguishable:

$$\left\{ (\mathbf{A}, \mathbf{A}^T\mathbf{s} + \mathbf{e}) \ : \ \begin{array}{l} \mathbf{A} \leftarrow \mathbb{Z}_2^{n \times m}, \\ \mathbf{s} \leftarrow \mathbb{Z}_2^n, \mathbf{e} \leftarrow \chi_{m,p}^{(\mathsf{e})} \end{array} \right\} \approx_c \left\{ (\mathbf{A}, \mathbf{u}) \ : \ \begin{array}{l} \mathbf{A} \leftarrow \mathbb{Z}_2^{n \times m}, \\ \mathbf{u} \leftarrow \mathbb{Z}_2^m \end{array} \right\}$$

[JKPT12] pointed out that the "sample-preserving reduction" from search to decision version of LPN of [AIK09, Lemma 4.4] holds for xLPN as well. Additionally, they pointed out that the search xLPN and search LPN are equivalent. Combining the above two facts, we know that the decision xLPN assumption holds *iff* decision LPN assumption holds.

Restricted-xLPN. In this work, we define a new version of the LPN problem which is based on the xLPN problem. We call it restricted-exact LPN (or rxLPN). This is defined exactly like the xLPN problem, except the size of the set of error vectors is a *power of two.*

More formally, let S denote the set $\chi_{m,p}^{(\mathsf{e})}$ and $t = |S|$. Also, let $\ell = \lfloor \log_2 t \rfloor$. We use $\chi_{m,p}^{(\mathsf{re})}$ to denote the subset of $\chi_{m,p}^{(\mathsf{e})}$ of size 2^ℓ consisting of lexically smallest elements. In other words, $\chi_{m,p}^{(\mathsf{re})}$ denotes the smallest 2^ℓ sized subset of $\chi_{m,p}^{(\mathsf{e})}$ as per the natural lexicographic ordering over integer sets. Concretely, $\chi_{m,p}^{(\mathsf{re})} = \left\{ S \subseteq \chi_{m,p}^{(\mathsf{e})} : |S| = 2^\ell \text{ and } \forall\ \mathbf{v} \in \chi_{m,p}^{(\mathsf{e})},\ \mathsf{int}(\mathbf{v}) > \max_{\mathbf{w} \in \chi_{m,p}^{(\mathsf{re})}} \mathsf{int}(\mathbf{w}) \vee \mathbf{v} \in \chi_{m,p}^{(\mathsf{re})} \right\}$.

Assumption 7 (Restricted Exact Learning Parity with Noise)**.** Let n, m be positive integers and p be a real number such that $p < 1/2$. The (Decision) Restricted Exact Learning Parity with Noise assumption $\mathsf{rxLPN}_{n,m,p}$, parameterized by the dimension of secret vector n, number of samples m and the error probability p, states that the following distributions are computationally indistinguishable:

$$\left\{ (\mathbf{A}, \mathbf{A}^T\mathbf{s} + \mathbf{e}) \ : \ \begin{array}{l} \mathbf{A} \leftarrow \mathbb{Z}_2^{n \times m}, \\ \mathbf{s} \leftarrow \mathbb{Z}_2^n, \mathbf{e} \leftarrow \chi_{m,p}^{(\mathsf{re})} \end{array} \right\} \approx_c \left\{ (\mathbf{A}, \mathbf{u}) \ : \ \begin{array}{l} \mathbf{A} \leftarrow \mathbb{Z}_2^{n \times m}, \\ \mathbf{u} \leftarrow \mathbb{Z}_2^m \end{array} \right\}$$

Equating rxLPN and LPN. Now we show that rxLPN is as hard as standard LPN. We start by making two important observations. First, we note that the "sample-preserving reduction" from *search to decision* version of LPN of [AIK09, Lemma 4.4] also holds for rxLPN. The sample-preserving reduction provided in Lemma 4.4 of [AIK09] simply uses the fact that by Goldreich-Levin hardcore bit theorem [GL89], given $(\mathbf{A}, \mathbf{A}^T\mathbf{s}{+}\mathbf{e})$ and a random n-bit vector $\mathbf{r}$, an efficient adversary cannot compute $\langle \mathbf{s}, \mathbf{r} \rangle$ with probability greater than $\frac{1}{2} + \text{negl}(n)$. To argue that *decision*-LPN is hard, they assume towards contradiction that suppose an efficient distinguisher exists. Next, they use the distinguisher to construct a hardcore-bit predictor for the underlying code (i.e., $(\mathbf{A}, \mathbf{A}^T\mathbf{s} + \mathbf{e})$). To complete the argument they show that if the distinguisher has non-negligible advantage, then the predictor will predict the hardcore-bit with probability greater than $\frac{1}{2}$ by a non-negligible amount. It turns out that the reduction provided in [AIK09] is independent of the choice of underlying error distribution. Therefore, using the same analysis, we get that *search* and *decision* variants of rxLPN are equivalent (up to a polynomial loss in the adversary's advantage).

Second, we note that the *search*-rxLPN and *search*-xLPN are equivalent. This is because we know that $|\chi_{m,p}^{(\mathsf{re})}| \geq |\chi_{m,p}^{(\mathsf{e})}|/2$. Therefore, if there exists an efficient adversary $\mathcal{A}$ that outputs the secret vector $\mathbf{s}$ (with non-negligible probability δ) given an instance of *search*-rxLPN problem, then we know that the same adversary $\mathcal{A}$ must output the secret vector $\mathbf{s}$ (with probability at least $\delta/2$) given an instance of *search*-xLPN problem; as with probability at least $\frac{1}{2}$, a random *search*-xLPN instance will also be a *search*-rxLPN instance. Thus, combining the above two facts, we get that *decision*-rxLPN and *search*-xLPN are equivalent.

Now recall that Jain et al. [JKPT12] pointed out that *search*-xLPN and *search*-LPN are equivalent, and since we already know that *search* and *decision* variants of LPN are equivalent, therefore combining all the above facts, we get that the *decision*-rxLPN and *decision*-LPN assumption are also equivalent. Thus, we get that rxLPN is as hard as standard LPN.[7] In the sequel, we will directly assume that decision-rxLPN is hard.

2.5 Injective Pseudorandom Generators with Setup

We will be considering PRGs with an additional setup algorithm that outputs public parameters. The setup algorithm will be important for achieving injectivity in our constructions. While this is weaker than the usual notion of PRGs (without setup), it turns out that for many of the applications that require injectivity of PRG, the setup phase is not an issue.

$\mathsf{Setup}(1^\lambda)$: The setup algorithm takes as input the security parameter λ and outputs public parameters pp, domain $\mathcal{D}$ and co-domain $\mathcal{R}$ of the PRG. Let params denote $(\mathsf{pp}, \mathcal{D}, \mathcal{R})$.

$\mathrm{PRG}(\mathsf{params}, s \in \mathcal{D})$: The PRG evaluation algorithm takes as input the public parameters and the PRG seed $s \in \mathcal{D}$, and outputs $y \in \mathcal{R}$.

Perfect Injectivity. A pseudorandom generator with setup $(\mathsf{Setup}, \mathrm{PRG})$ is said to have perfect injectivity if for all $(\mathsf{pp}, \mathcal{D}, \mathcal{R}) \leftarrow \mathsf{Setup}(1^\lambda)$, for all $s_1 \neq s_2 \in \mathcal{D}$, $\mathrm{PRG}(\mathsf{params}, s_1) \neq \mathrm{PRG}(\mathsf{params}, s_2)$.

Pseudorandomness. A pseudorandom generator with setup $(\mathsf{Setup}, \mathrm{PRG})$ is said to be secure if for any PPT adversary $\mathcal{A}$, there exists a negligible function $\mathrm{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\Pr\left[\mathcal{A}(\mathsf{params}, t_b) = b \;:\; \begin{array}{c} \mathsf{params} \leftarrow \mathsf{Setup}(1^\lambda) \\ s \leftarrow \mathcal{D}, t_0 \leftarrow \mathcal{R}, b \leftarrow \{0,1\} \\ t_1 = \mathrm{PRG}(\mathsf{params}, s) \end{array}\right] \leq \frac{1}{2} + \mathrm{negl}(\lambda).$$

2.6 Lockable Obfuscation

In this section, we recall the notion of lockable obfuscation defined by Goyal et al. [GKW17a]. Let n, m, d be polynomials, and $\mathcal{C}_{n,m,d}(\lambda)$ be the class of depth $d(\lambda)$ circuits with $n(\lambda)$ bit input and $m(\lambda)$ bit output. Let $\mathcal{M}$ be the message space. A lockable obfuscator for $\mathcal{C}_{n,m,d}$ consists of algorithms Obf and Eval with the following syntax.

- $\mathsf{Obf}(1^\lambda, P, \mathsf{msg}, \alpha) \to \widetilde{P}$. The obfuscation algorithm is a randomized algorithm that takes as input the security parameter λ, a program $P \in \mathcal{C}_{n,m,d}$, message $\mathsf{msg} \in \mathcal{M}$ and 'lock string' $\alpha \in \{0,1\}^{m(\lambda)}$. It outputs a program $\widetilde{P}$.
- $\mathsf{Eval}(\widetilde{P}, x) \to y \in \mathcal{M} \cup \{\perp\}$. The evaluator is a deterministic algorithm that takes as input a program $\widetilde{P}$ and a string $x \in \{0,1\}^{n(\lambda)}$. It outputs $y \in \mathcal{M} \cup \{\perp\}$.

Correctness For correctness, we require that if $P(x) = \alpha$, then the obfuscated program $\widetilde{P} \leftarrow \mathsf{Obf}(1^\lambda, P, \mathsf{msg}, \alpha)$, evaluated on input x, outputs msg, and if $P(x) \neq \alpha$, then $\widetilde{P}$ outputs $\perp$ on input x. Formally,

Definition 2.1 (Perfect Correctness)**.** Let n, m, d be polynomials. A lockable obfuscation scheme for $C_{n,m,d}$ and message space $\mathcal{M}$ is said to be perfectly correct if it satisfies the following properties:

[7] At first sight it might seem that we might be able to attack these restricted notions of LPN by using results such as [AG11], since the corresponding noise distributions are very well structured. However, Arora-Ge [AG11] attack does not apply here, as for their attack to work the noise vector should be sampled from a special distribution where the vector is divided into blocks of suitable size, and in each block, there are a bounded number of 1s. And if each block has p bits with at most w 1s, then they show how to extract the secret in time $O(p^w)$. In the exact-LPN and restricted-exact LPN assumptions, the blocks have size polynomial in the security parameter, and the number of 1s is $O(\sqrt{n})$. Thus, the attack does not work.

1. For all security parameters λ, inputs $x \in \{0,1\}^{n(\lambda)}$, programs $P \in \mathcal{C}_{n,m,d}$ and messages $\mathsf{msg} \in \mathcal{M}$, if $P(x) = \alpha$, then
$$\mathsf{Eval}(\mathsf{Obf}(1^\lambda, P, \mathsf{msg}, \alpha), x) = \mathsf{msg}.$$

2. For all security parameters λ, inputs $x \in \{0,1\}^{n(\lambda)}$, programs $P \in \mathcal{C}_{n,m,d}$ and messages $\mathsf{msg} \in \mathcal{M}$, if $P(x) \neq \alpha$, then
$$\mathsf{Eval}(\mathsf{Obf}(1^\lambda, P, \mathsf{msg}, \alpha), x) = \perp .$$

Remark 2.1 (Weaker notions of correctness)**.** We would like to point out that GKW additionally defined two weaker notions of correctness - statistical and semi-statistical correctness. They say that lockable obfuscation satisfies statistical correctness if for any triple $(P, \mathsf{msg}, \alpha)$, the probability that there exists an x s.t. $P(x) \neq \alpha$ and the obfuscated program outputs msg on input x is negligible in security parameter. The notion of semi-statistical correctness is even weaker where each obfuscated program could potentially always output message msg for some input x s.t. $P(x) \neq \alpha$, but if one fixes the input x before obfuscation, then the probability of the obfuscated program outputting msg on input x is negligible.

Security We now present the simulation based security definition for Lockable Obfuscation.

Definition 2.2. Let n, m, d be polynomials. A lockable obfuscation scheme $(\mathsf{Obf}, \mathsf{Eval})$ for $\mathcal{C}_{n,m,d}$ and message space $\mathcal{M}$ is said to be secure if there exists a PPT simulator Sim such that for all PPT adversaries $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$, there exists a negligible function negl$(\cdot)$ such that the following function is bounded by negl$(\cdot)$:

$$\left| \Pr\left[\mathcal{A}_1(\tilde{P_b}, \mathsf{st}) = b : \begin{array}{c} (P \in \mathcal{C}_{n,m,d}, \mathsf{msg} \in \mathcal{M}, \mathsf{st}) \leftarrow \mathcal{A}_0(1^\lambda) \\ b \leftarrow \{0,1\}, \alpha \leftarrow \{0,1\}^{m(\lambda)} \\ \tilde{P_0} \leftarrow \mathsf{Obf}(1^\lambda, P, \mathsf{msg}, \alpha) \\ \tilde{P_1} \leftarrow \mathsf{Sim}(1^\lambda, 1^{|P|}, 1^{|\mathsf{msg}|}) \end{array} \right] - \frac{1}{2} \right|$$

3 Perfectly Injective PRGs from LWR

In this construction, we will present a construction based on the Learning With Rounding (LWR) assumption. For any two moduli $2 \leq p < q$ and integer x in $\mathbb{Z}_q$, let $\lfloor x \rceil_p$ denote $\lfloor (p/q) \cdot x \rceil$. At a high level, the construction works as follows: the setup algorithm chooses a uniformly random matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times 2m}$, where m is much greater than n. The PRG evaluation outputs $\lfloor \mathbf{x}^T \cdot \mathbf{A} \rceil_p$, where $p = 2^{\ell_{\text{out}}}$. Note that this already gives us a PRG with statistical injectivity. However, to achieve perfect injectivity, we need to ensure that the matrix $\mathbf{A}$ is full rank, and that injectivity is preserved even after rounding. In order to achieve this, we need to make some modifications to the setup algorithm.

The new setup algorithm chooses a uniformly random matrix $\mathbf{B}$, a random matrix $\mathbf{R}$ with ± 1 entries. Let $\mathbf{D}$ be a fixed full rank matrix with 'medium sized' entries. It then outputs $\mathbf{A} = [\mathbf{B} \mid \mathbf{BR} + \mathbf{D}]$. The PRG evaluation is same as described above.

We will now describe the algorithms formally.

$\mathsf{Setup}(1^\lambda)$ The setup algorithm first sets the parameters $n, m, q, \ell_{\text{out}}, \rho$ in terms of the security parameter. These parameters must satisfy the following constraints.

- $n = \mathsf{poly}(\lambda)$
- $q \leq 2^{n^\epsilon}$
- $m > 2n \log q$
- $p = 2^{\ell_{\text{out}}}$
- $n < m \cdot \ell_{\text{out}}$
- $(q/p)m < \rho < q$

One particular setting of parameters which satisfies the constraints above is as follows: set $n = \mathsf{poly}(\lambda)$, $q = 2^{n^{\epsilon}}$, $p = \sqrt{q}$, $m = n^2$ and $\rho = q/4$.

Next, it chooses a matrix $\mathbf{B} \leftarrow \mathbb{Z}_q^{n \times m}$, matrix $\mathbf{R} \leftarrow \{+1,-1\}^{m \times m}$. Let $\mathbf{D} = \rho \cdot [\mathbf{I}_n \mid \mathbf{0}^{n \times (m-n)}]$ and $\mathbf{A} = [\mathbf{B} \mid \mathbf{B} \cdot \mathbf{R} + \mathbf{D}]$. The setup algorithm outputs $\mathbf{A}$ as the public parameters. It sets the domain $\mathcal{D} = \{0,1\}^n$ and co-domain $\mathcal{R} = \{0,1\}^{m \cdot \ell_{\text{out}}}$.

PRG$(\mathbf{A}, \mathbf{s})$: The PRG evaluation algorithm takes as input the matrix $\mathbf{A}$ and the seed $\mathbf{s} \in \{0,1\}^n$. It computes $\mathbf{y} = \mathbf{s}^T \cdot \mathbf{A}$. Finally, it outputs $\lfloor \mathbf{y} \rceil_p \in \mathbb{Z}_p^m$ as a bit string of length $2m \cdot \ell_{\text{out}}$.

Depth of PRG **Evaluation Circuit and** PRG **Stretch.** First, note that the the PRG evaluation circuit only needs to perform a single matrix-vector multiplication followed by discarding the $\lceil \log_2 q/p \rceil$ least significant bits of each element. Clearly such a circuit can be implemented in $\mathbf{TC}^0$, the class of constant-depth, poly-sized circuits with unbounded fan-in and threshold gates (which is a subset of $\mathbf{NC}^1$). Additionally, the stretch provided by the above PRG could be arbitrarily set during setup. Thus, the above construction gives a PRG that provides a polynomial stretch with a $\mathbf{TC}^0$ evaluation circuit.

Next, we prove the following theorem where we first show that our PRG construction satisfies perfect injectivity property, and later argue the pseudorandomness property for the same.

Theorem 3.1. If the LWR assumption with parameters n, m, p and q (Assumption 3) holds, then the above construction is a perfectly injective PRG.

3.1 Perfect Injectiveness

Let $\mathbf{A} = [\mathbf{B} \mid \mathbf{B} \cdot \mathbf{R} + \mathbf{D}]$ be the matrix output by the setup algorithm, and let $\mathbf{s}, \mathbf{s}' \in \{0,1\}^n$ be two strings. Let $\mathbf{s}^T \cdot \mathbf{A} = [\mathbf{z}_1 \mid \mathbf{z}_2]$ where $\mathbf{z}_1, \mathbf{z_2} \in \mathbb{Z}_q^m$, and let $\mathbf{y}_j = \lfloor \mathbf{z}_j \rceil_p$ for $j \in \{1,2\}$. Similarly, $\mathbf{s}'^T \cdot \mathbf{A} = [\mathbf{z}'_1 \mid \mathbf{z}'_2]$ and $\mathbf{y}'_j = \lfloor \mathbf{z}'_j \rceil_p$ for $j \in \{1,2\}$.

Suppose $\mathbf{y}_1 = \mathbf{y}'_1$ and $\mathbf{y}_2 = \mathbf{y}'_2$. We will show that $\mathbf{s} = \mathbf{s}'$. Let $\mathbf{w} = \lfloor \mathbf{s}^T \cdot \mathbf{B} \cdot \mathbf{R} \rceil_p$ and $\mathbf{w}' = \lfloor \mathbf{s}'^T \cdot \mathbf{B} \cdot \mathbf{R} \rceil_p$.

Lemma 3.1. If $\mathbf{y}_1 = \mathbf{y}'_1$, then for every index $j \in \{1, 2, \ldots, m\}$, $\left| (\mathbf{w} - \mathbf{w}')_j \mod p \right| \le m$.

Proof. Since $\mathbf{y}_1 = \mathbf{y}'_1$, for every index $j \in \{1, 2, \ldots, m\}$, $\left| (\mathbf{z}_1 - \mathbf{z}'_1)_j \mod q \right| \le (q/p)$. As a result, for any $\mathbf{r} \in \{+1,-1\}^m$, $\left| \left(\mathbf{z}_1^T \cdot \mathbf{r} - \mathbf{z}_1'^T \cdot \mathbf{r} \right) \mod q \right| \le (q/p) \cdot m$. Extending this argument, for any matrix $\mathbf{R} \in \{+1,-1\}^{m \times m}$ and any index $j \in \{1, 2, \ldots, m\}$, $\left| \left(\mathbf{z}_1^T \cdot \mathbf{R} - \mathbf{z}_1'^T \cdot \mathbf{R} \right)_j \mod q \right| \le (q/p) \cdot m$. Therefore,

$$\begin{aligned} \left| (\mathbf{w} - \mathbf{w})'_j \mod p \right| &= \left| \lfloor \left(\mathbf{z}_1^T \cdot \mathbf{R} \right)_j \rceil_p - \lfloor \left(\mathbf{z}_1'^T \cdot \mathbf{R} \right)_j \rceil_p \mod p \right| \\ &= \left| \lfloor \frac{p}{q} \left(\mathbf{z}_1^T \cdot \mathbf{R} \right)_j \rceil - \lfloor \frac{p}{q} \left(\mathbf{z}_1'^T \cdot \mathbf{R} \right)_j \rceil \mod p \right| \le m \end{aligned}$$

■

Since, for all j, $\left| (\mathbf{w} - \mathbf{w}')_j \mod p \right| \le m$ and $\mathbf{y}_2 = \mathbf{y}'_2$, $\left| \left(\lfloor \mathbf{s}^T \cdot \mathbf{D} \rceil_p - \lfloor \mathbf{s}'^T \cdot \mathbf{D} \rceil_p \right)_j \mod p \right| \le m$, and therefore $\left| \lfloor \rho \cdot \mathbf{s}_j \rceil_p - \lfloor \rho \cdot \mathbf{s}'_j \rceil_p \mod p \right| \le m$. Since $(q/p) \cdot m < \rho < q$ and $\mathbf{s}, \mathbf{s}'$ are bit vectors, it follows that $\left| \lfloor \rho \cdot \mathbf{s}_j \rceil_p - \lfloor \rho \cdot \mathbf{s}'_j \rceil_p \mod p \right| \le m$ if and only if $\mathbf{s} = \mathbf{s}'$.

3.2 Pseudorandomness

In order to prove pseudorandomness, we will define a sequence of hybrid experiments. First, we will switch the matrix $\mathbf{A}$ output by the setup algorithm to a uniformly random matrix. This step is information theoretic (due to Leftover Hash Lemma). Then, we can use the LWR assumption to argue that $\lfloor \mathbf{s}^T \cdot \mathbf{B} \rceil_p$ is indistinguishable from a uniformly random vector in $\mathbb{Z}_p^{2m}$.

Hybrid H_0 This corresponds to the real experiment.

1. The challenger chooses a uniformly random matrix $\mathbf{B} \leftarrow \mathbb{Z}_q^{n\times m}$, a uniformly random matrix $\mathbf{R} \leftarrow \{+1,-1\}^{m\times m}$ and $\mathbf{D} = \rho \cdot \left[\mathbf{I}_n \mid \mathbf{0}^{n\times(m-n)}\right]$. It sets $\mathbf{A} = [\mathbf{B} \mid \mathbf{B}\cdot\mathbf{R} + \mathbf{D}]$ and sends it to the adversary.
2. Next, the challenger chooses a uniformly random bit-string $\mathbf{s} \leftarrow \{0,1\}^n$ and sets $\mathbf{y}_0 = \lfloor \mathbf{s}^T \cdot \mathbf{A} \rceil_p$. It also chooses $\mathbf{y}_1 \leftarrow \mathbb{Z}_p^m$ and bit $b \leftarrow \{0,1\}$. The challenger sends $\mathbf{y}_b$ to $\mathcal{A}$.
3. The adversary sends a bit b' and wins if $b = b'$.

Hybrid H_1 In this experiment, the challenger chooses the matrix $\mathbf{A}$ uniformly at random from $\mathbb{Z}_q^{n\times 2m}$.

1. The challenger chooses a uniformly random matrix $\mathbf{A} \leftarrow \mathbb{Z}_q^{n\times 2m}$ and sends it to the adversary.
2. Next, the challenger chooses a uniformly random bit-string $\mathbf{s} \leftarrow \{0,1\}^n$ and sets $\mathbf{y}_0 = \lfloor \mathbf{s}^T \cdot \mathbf{A} \rceil_p$. It also chooses $\mathbf{y}_1 \leftarrow \mathbb{Z}_p^m$ and bit $b \leftarrow \{0,1\}$. The challenger sends $\mathbf{y}_b$ to $\mathcal{A}$.
3. The adversary sends a bit b' and wins if $b = b'$.

Hybrid H_2 In this experiment, the challenger sets the output string to a uniformly random string.

1. The challenger chooses a uniformly random matrix $\mathbf{A} \leftarrow \mathbb{Z}_q^{n\times 2m}$ and sends it to the adversary.
2. Next, the challenger chooses a uniformly random bit-string $\mathbf{y} \leftarrow \{0,1\}^{m\cdot\ell_{\mathrm{out}}}$ and outputs $\mathbf{y}$.
3. The adversary sends a bit b' and wins if $b = b'$.

Analysis Let $\mathsf{Adv}_i^{\mathcal{A}}$ denote the advantage of adversary $\mathcal{A}$ in Hybrid H_i.

Lemma 3.2. For any adversary $\mathcal{A}$, $|\mathsf{Adv}_0^{\mathcal{A}} - \mathsf{Adv}_1^{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. Note that $\mathbf{R} \leftarrow \{+1,-1\}^{m\times m}$ and $\mathbf{H}_\infty(\mathbf{R}) = m^2$ (min-entropy of $\mathcal{R}$). As $m^2 = n \cdot m \cdot \log_2 q + \omega(\log n)$, it follows from Leftover Hash Lemma (Corollary 2.1) that the following distributions are statistically indistinguishable:

$$\{(\mathbf{B}, \mathbf{B}\cdot\mathbf{R}) \ : \ \mathbf{B} \leftarrow \mathbb{Z}_q^{n\times m}, \mathbf{R} \leftarrow \{+1,-1\}^{m\times m}\} \approx_s \{(\mathbf{B}, \mathbf{U}) \ : \ \mathbf{B} \leftarrow \mathbb{Z}_q^{n\times m}, \mathbf{U} \leftarrow \mathbb{Z}_q^{n\times m}\}$$

As a result, given any matrix $\mathbf{D}$, the matrix $\mathbf{A} = [\mathbf{B} \mid \mathbf{B}\cdot\mathbf{R} + \mathbf{D}]$ is statistically indistinguishable from a uniformly random matrix from $\mathbb{Z}_q^{n\times 2m}$. ■

Lemma 3.3. Assuming the Binary Learning with Rounding assumption with moduli q, p and dimension n, for any PPT adversary $\mathcal{A}$, $|\mathsf{Adv}_1^{\mathcal{A}} - \mathsf{Adv}_2^{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ such that $|\mathsf{Adv}_1^{\mathcal{A}} - \mathsf{Adv}_2^{\mathcal{A}}| = \epsilon$. Then there exists a PPT reduction algorithm $\mathcal{B}$ that can break the Binary LWR assumption with advantage ϵ.

The reduction algorithm receives $\mathbf{A} \in \mathbb{Z}_q^{n\times 2m}, \mathbf{y} \in \mathbb{Z}_p^m$ from the LWR challenger, which it forwards to the PRG adversary. The adversary outputs a bit b', which the reduction algorithm forwards to the LWR challenger. Clearly, if $\mathcal{A}$ wins with advantage ϵ in the PRG game, then $\mathcal{B}$ breaks the LWR assumption with advantage ϵ. ■

Finally, note that any adversary has 0 advantage in the hybrid H_2. From the above lemmas, it follows that under the LWR assumption, the PRG construction is secure.

4 Lockable Obfuscation with Perfect Correctness

4.1 Construction

In this section, we present our perfectly correct lockable obfuscation scheme. We note that the construction is similar to the statistically correct lockable obfuscation scheme described in Goyal et al. [GKW17a]. A part of the description has been taken verbatim from [GKW17a]. For any polynomials $\ell_{\text{in}}, \ell_{\text{out}}, d$ such that $\ell_{\text{out}} = \omega(\log \lambda)$, we construct a lockable obfuscation scheme $\mathcal{O} = (\mathsf{Obf}, \mathsf{Eval})$ for the circuit class $\mathcal{C}_{\ell_{\text{in}}, \ell_{\text{out}}, d}$. The message space for our construction will be $\{0,1\}$, although one can trivially extend it to $\{0,1\}^{\ell(\lambda)}$ for any polynomial ℓ [GKW17a].

The tools required for our construction are as follows:

- A compact leveled homomorphic bit encryption scheme $(\mathsf{LHE.Setup}, \mathsf{LHE.Enc}, \mathsf{LHE.Eval}, \mathsf{LHE.Dec})$ with decryption circuit of depth $d_{\mathsf{Dec}}(\lambda)$ and ciphertexts of length $\ell_{\mathsf{ct}}(\lambda)$.
- A *perfectly injective* pseudorandom generator scheme $(\text{PRG}.\mathsf{Setup}, \text{PRG}.\mathsf{Eval})$, where $\text{PRG}.\mathsf{Eval}$ has depth $d_{\text{PRG}}(\lambda)$, input length $\ell_{\text{out}}(\lambda)$ and output length $\ell_{\text{PRG}}(\lambda)$.

For notational convenience, let $\ell_{\text{in}} = \ell_{\text{in}}(\lambda)$, $\ell_{\text{out}} = \ell_{\text{out}}(\lambda)$, $\ell_{\text{PRG}} = \ell_{\text{PRG}}(\lambda)$, $d_{\mathsf{Dec}} = d_{\mathsf{Dec}}(\lambda)$, $d_{\text{PRG}} = d_{\text{PRG}}(\lambda)$ and $d = d(\lambda)$.

Fix any $\epsilon < 1/2$. Let χ be a B-bounded discrete Gaussian distribution with parameter σ such that $B = \sqrt{m} \cdot \sigma$. Let $n, m, \ell, \sigma, q, \mathsf{Bd}$ be parameters with the following constraints:

- $n = \mathsf{poly}(\lambda)$ and $q \leq 2^{n^{\epsilon}}$ (for LWE security)
- $m \geq \widetilde{c} \cdot n \cdot \log q$ for some universal constant $\widetilde{c}$ (for $\mathsf{SamplePre}$)
- $\sigma = \omega(\sqrt{n \cdot \log q \cdot \log m})$ (for Preimage Well Distributedness)
- $\ell_{\text{PRG}} = n \cdot m \cdot \log q + \omega(\log n)$ (for applying Leftover Hash Lemma)
- $\ell_{\text{PRG}} \cdot (L+1) \cdot (m^2 \cdot \sigma)^{L+1} < q^{1/8}$ (where $L = \ell_{\text{out}} \cdot \ell_{\mathsf{ct}} \cdot 4^{d_{\mathsf{Dec}} + d_{\text{PRG}}}$) (for correctness of scheme)

It is important that $L = \lambda^c$ for some constant c and $\ell_{\text{PRG}} \cdot (L+1) \cdot (m^2 \cdot \sigma)^{L+1} < q^{1/8}$. This crucially relies on the fact that the LHE scheme is compact (so that ℓ_{ct} and ℓ_{PRG} are bounded by a polynomial independent of the size of the circuits supported by the scheme, and that the LHE decryption and PRG computation can be performed by a log depth circuit (i.e, have poly length branching programs). The constant c depends on the LHE scheme and PRG.

One possible setting of parameters is as follows: $n = \lambda^{4c/\epsilon}$, $m = n^{1+2\epsilon}$, $q = 2^{n^{\epsilon}}$, $\sigma = n$ and $\ell_{\text{PRG}} = n^{3\epsilon+3}$. We will now describe the obfuscation and evaluation algorithms.

- $\mathsf{Obf}(1^{\lambda}, P, \mathsf{msg}, \alpha)$: The obfuscation algorithm takes as input a program $P \in \mathcal{C}_{\ell_{\text{in}}, \ell_{\text{out}}, d}$, message $\mathsf{msg} \in \{0,1\}$ and $\alpha \in \{0,1\}^{\ell_{\text{out}}}$. The obfuscator proceeds as follows:
  1. First, it chooses the LHE key pair as $(\mathsf{lhe.sk}, \mathsf{lhe.ek}) \leftarrow \mathsf{LHE.Setup}(1^{\lambda}, 1^{d \log d})$.[8]
  2. Next, it encrypts the program P. It sets $\mathbf{ct} \leftarrow \mathsf{LHE.Enc}(\mathsf{lhe.sk}, P)$.[9]
  3. It runs $\mathsf{pp} \leftarrow \text{PRG}.\mathsf{Setup}(1^{\lambda})$, and assigns $\beta = \text{PRG}.\mathsf{Eval}(\mathsf{pp}, \alpha)$.
  4. Next, consider the following circuit Q which takes as input $\ell_{\text{out}} \cdot \ell_{\mathsf{ct}}$ bits of input and outputs ℓ_{PRG} bits. Q takes as input ℓ_{out} LHE ciphertexts $\{\mathsf{ct}_i\}_{i \leq \ell_{\text{out}}}$, has LHE secret key $\mathsf{lhe.sk}$ hardwired and computes the following — (1) it decrypts each input ciphertext ct_i (in parallel) to get string x of length ℓ_{out} bits, (2) it applies the PRG on x and outputs $\text{PRG}.\mathsf{Eval}(\mathsf{pp}, x)$. Concretely, $Q(\mathsf{ct}_1, \ldots, \mathsf{ct}_{\ell_{\text{out}}}) = \text{PRG}.\mathsf{Eval}\big(\mathsf{pp}, \mathsf{LHE.Dec}(\mathsf{lhe.sk}, \mathsf{ct}_1) \,||\, \cdots \,||\, \mathsf{LHE.Dec}(\mathsf{lhe.sk}, \mathsf{ct}_{\ell_{\text{out}}})\big)$.

[8] We set the LHE depth bound to be $d \log d$, where the extra log factor is to account for the constant blowup involved in using a universal circuit. In particular, we can set the LHE depth bound to be $c \cdot d$ where c is some fixed constant depending on the universal circuit.

[9] Note that LHE scheme supports bit encryption. Therefore, to encrypt P, a multi-bit message, the $\mathsf{FHE.Enc}$ algorithm will be run independently on each bit of P. However, for notational convenience throughout this section we overload the notation and use $\mathsf{FHE.Enc}$ and $\mathsf{FHE.Dec}$ algorithms to encrypt and decrypt multi-bit messages respectively.

For $i \leq \ell_{\text{PRG}}$, we use $\mathsf{BP}^{(i)}$ to denote the fixed-input selector permutation branching program that outputs the i^{th} bit of output of circuit Q. Note that Q has depth $d_{\mathsf{tot}} = d_{\mathsf{Dec}} + d_{\text{PRG}}$. By Corollary B.1, we know that each branching program $\mathsf{BP}^{(i)}$ has length $L = \ell_{\text{out}} \cdot \ell_{\mathsf{ct}} \cdot 4^{d_{\mathsf{tot}}}$ and width 5.

5. Finally, the obfuscator creates matrix components which enable the evaluator to compute msg if it has an input strings (ciphertexts) $\mathsf{ct}_1, \ldots, \mathsf{ct}_{\ell_{\text{out}}}$ such that $Q(\mathsf{ct}_1, \ldots, \mathsf{ct}_{\ell_{\text{out}}}) = \beta$. Concretely, it runs the (randomized) routine $\mathsf{Comp\text{-}Gen}$ (defined in Figure 1). This routine takes as input the circuit Q in the form of ℓ_{PRG} branching programs $\{\mathsf{BP}^{(i)}\}_i$, string β and message msg. Let $\left(\left\{\mathbf{B}^{(i)}_{0,1}\right\}_i, \left\{\mathbf{C}^{(i,0)}_j, \mathbf{C}^{(i,1)}_j\right\}_{i,j}\right) \leftarrow \mathsf{Comp\text{-}Gen}(\{\mathsf{BP}^{(i)}\}_i, \beta, \mathsf{msg})$.
6. The final obfuscated program consists of the LHE evaluation key $\mathsf{ek} = \mathsf{lhe.ek}$, LHE ciphertexts $\mathbf{ct}$, together with the components $\left(\left\{\mathbf{B}^{(i)}_{0,1}\right\}_i, \left\{(\mathbf{C}^{(i,0)}_j, \mathbf{C}^{(i,1)}_j)\right\}_{i,j}\right)$.

- $\mathsf{Eval}(\tilde{P}, x)$: The evaluation algorithm takes as input $\tilde{P} = \left(\mathsf{ek}, \mathbf{ct}, \left\{\mathbf{B}^{(i)}_{0,1}\right\}_i, \left\{(\mathbf{C}^{(i,0)}_j, \mathbf{C}^{(i,1)}_j)\right\}_{i,j}\right)$ and input $x \in \{0,1\}^{\ell_{\text{in}}}$. It performs the following steps.

  1. The evaluator first constructs a universal circuit $U_x(\cdot)$ with x hardwired as input. This universal circuit takes a circuit C as input and outputs $U_x(C) = C(x)$. Using the universal circuit of Cook and Hoover [CH85], it follows that $U_x(\cdot)$ has depth $O(d)$.
  2. Next, it performs homomorphic evaluation on $\mathbf{ct}$ using circuit $U_x(\cdot)$. It computes $\widetilde{\mathbf{ct}} = \mathsf{LHE.Eval}(\mathsf{ek}, U_x(\cdot), \mathbf{ct})$. Note that $\ell_{\mathsf{ct}} \cdot \ell_{\text{out}}$ denotes the length of $\widetilde{\mathbf{ct}}$ (as a bitstring), and let $\widetilde{\mathbf{ct}}_i$ denote the i^{th} bit of $\widetilde{\mathbf{ct}}$.
  3. The evaluator then obliviously evaluates the ℓ_{PRG} branching programs on input $\widetilde{\mathbf{ct}}$ using the matrix components. It calls the component evaluation algorithm $\mathsf{Comp\text{-}Eval}$ (defined in Figure 2). Let $y = \mathsf{Comp\text{-}Eval}\left(\widetilde{\mathbf{ct}}, \left(\left\{\mathbf{B}^{(i)}_{0,1}\right\}_i, \left\{(\mathbf{C}^{(i,0)}_j, \mathbf{C}^{(i,1)}_j)\right\}_{i,j}\right)\right)$. The evaluator outputs y.

4.2 Correctness

We will prove that the lockable obfuscation scheme described above satisfies the perfect correctness property (see 2.1). To prove this, we need to prove that if $P(x) = \alpha$, then the evaluation algorithm always outputs the message, and if $P(x) \neq \alpha$, then it always outputs $\perp$.

First, we will prove the following lemma about the $\mathsf{Comp\text{-}Gen}$ and $\mathsf{Comp\text{-}Eval}$ routines. For any $z \in \{0,1\}^{\ell_{\text{in}}(\lambda)}$, let $\mathsf{BP}(z) = \mathsf{BP}^{(1)}(z) \,||\, \mathsf{BP}^{(2)}(z) \,||\, \ldots \,||\, \mathsf{BP}^{(\ell_{\text{PRG}})}(z)$. Intuitively, this lemma states that for all fixed input branching programs $\{\mathsf{BP}^{(i)}\}_i$, strings β, input z, and messages msg, if $\mathsf{BP}(z) = \beta$, then the component evaluator outputs msg.

Lemma 4.1. For any set of branching programs $\{\mathsf{BP}^{(i)}\}_{i \leq \ell_{\text{PRG}}}$, string $\beta \in \{0,1\}^{\ell_{\text{PRG}}}$, message $\mathsf{msg} \in \{0,1\}$ and input z,

1. if $\mathsf{BP}(z) = \beta$, then $\mathsf{Comp\text{-}Eval}(z, \mathsf{Comp\text{-}Gen}(\{\mathsf{BP}^{(i)}\}_i, \beta, \mathsf{msg})) = \mathsf{msg}$.
2. if $\mathsf{BP}(z) \neq \beta$, then $\mathsf{Comp\text{-}Eval}(z, \mathsf{Comp\text{-}Gen}(\{\mathsf{BP}^{(i)}\}_i, \beta, \mathsf{msg})) = \perp$.

Proof. Recall that the component generation algorithm chooses matrices $\mathbf{B}^{(i)}_j$ for each $i \leq \ell_{\text{PRG}}$, $j \leq L$, $\mathbf{S}^{(0)}_j, \mathbf{S}^{(1)}_j$ for each $j \leq L$ and $\mathbf{E}^{(i,0)}_j, \mathbf{E}^{(i,1)}_j$ for each $i \leq \ell_{\text{PRG}}, j \leq L$. Note that the $\mathbf{S}^{(b)}_j$ and $\mathbf{E}^{(i,b)}_j$ matrices have l_∞ norm bounded by $\sigma \cdot m^{3/2}$ since they are chosen from truncated Gaussian distribution with parameter σ.

We start by introducing some notations for this proof.

- $\mathsf{st}^{(i)}_j$: the state of $\mathsf{BP}^{(i)}$ after j steps when evaluated on z

Comp-Gen

Input: $\{\mathsf{BP}^{(i)}\}_i$, $\beta \in \{0,1\}^{\ell_{\mathrm{PRG}}}$, $\mathsf{msg} \in \{0,1\}$

Output: Components $\left(\left\{\mathbf{B}_{0,1}^{(i)}\right\}_i, \{(\mathbf{C}_{\mathsf{level}}^{(i,0)}, \mathbf{C}_{\mathsf{level}}^{(i,1)})\}_{i \leq \ell_{\mathrm{PRG}}, \mathsf{level} \leq L}\right)$.

(a) Let $\mathsf{BP}^{(i)} = \left(\left\{\sigma_{j,b}^{(i)} : [5] \to [5]\right\}_{j \in [L], b \in \{0,1\}}, \mathsf{acc}^{(i)} \in [5], \mathsf{rej}^{(i)} \in [5]\right)$ for all $i \leq \ell_{\mathrm{PRG}}$.

(b) First, it chooses a matrix for each state of each branching program. Recall, there are ℓ_{PRG} branching programs, and each branching program has L levels, and each level has 5 states. For each $i \leq \ell_{\mathrm{PRG}}$, $j \in [0, L-1]$, it chooses a matrix of dimensions $5n \times m$ along with its trapdoors (independently) as $(\mathbf{B}_j^{(i)}, T_j^{(i)}) \leftarrow \mathsf{TrapGen}(1^{5n}, 1^m, q)$. The matrix $\mathbf{B}_j^{(i)}$ can be parsed as follows

$$\mathbf{B}_j^{(i)} = \begin{bmatrix} \mathbf{B}_{j,1}^{(i)} \\ \vdots \\ \mathbf{B}_{j,5}^{(i)} \end{bmatrix}$$

where matrices $\mathbf{B}_{j,k}^{(i)} \in \mathbb{Z}_q^{n \times m}$ for $k \leq 5$. The matrix $\mathbf{B}_{j,k}^{(i)}$ corresponds to state k at level j of branching program $\mathsf{BP}^{(i)}$.

(c) Let $\mathbf{D} = q^{3/4} \cdot \left[\mathbf{I}_n \,||\, \mathbf{0}^{n \times (m - 2 \cdot n)}\right]$. For the top level, it first chooses the matrices $\mathbf{A}_{L,k}^{(i)}$ (of dimension $n \times n$) for each $i \leq \ell_{\mathrm{PRG}}, k \leq 5$, uniformly at random, subject to the following constraint:

$$\sum_{i : \beta_i = 0} \mathbf{A}_{L,\mathsf{rej}^{(i)}}^{(i)} + \sum_{i : \beta_i = 1} \mathbf{A}_{L,\mathsf{acc}^{(i)}}^{(i)} = \mathbf{0}^{n \times n} \text{ if } \mathsf{msg} = 0.$$

$$\sum_{i : \beta_i = 0} \mathbf{A}_{L,\mathsf{rej}^{(i)}}^{(i)} + \sum_{i : \beta_i = 1} \mathbf{A}_{L,\mathsf{acc}^{(i)}}^{(i)} = q^{1/4} \cdot \mathbf{I}_n \text{ if } \mathsf{msg} = 1.$$

It then samples a matrix $\mathbf{S} \leftarrow \chi^{n \times (m-n)}$, and matrices $\mathbf{E}_{L,\mathsf{rej}^{(i)}}^{(i)} \leftarrow \chi^{n \times (m-n)}, \mathbf{E}_{L,\mathsf{acc}^{(i)}}^{(i)} \leftarrow \chi^{n \times (m-n)}$ for each $i \leq \ell_{\mathrm{PRG}}$. It then chooses matrices $\mathbf{F}_{L,k}^{(i)}$ as follows

$$\mathbf{F}_{L,\mathsf{acc}^{(i)}}^{(i)} = \mathbf{A}_{L,\mathsf{acc}^{(i)}}^{(i)} \cdot \mathbf{S} + \mathbf{E}_{L,\mathsf{acc}^{(i)}}^{(i)} + (1 - \beta_i) \cdot \mathbf{D}, \qquad \mathbf{F}_{L,\mathsf{rej}^{(i)}}^{(i)} = \mathbf{A}_{L,\mathsf{rej}^{(i)}}^{(i)} \cdot \mathbf{S} + \mathbf{E}_{L,\mathsf{rej}^{(i)}}^{(i)} + \beta_i \cdot \mathbf{D}$$

$$\mathbf{F}_{L,k}^{(i)} \leftarrow \mathbb{Z}_q^{n \times (m-n)} \text{ if } k \notin \{\mathsf{acc}^{(i)}, \mathsf{rej}^{(i)}\}$$

The top level matrices $\mathbf{B}_{L,k}^{(i)}$ for each $i \leq \ell_{\mathrm{PRG}}, k \leq 5$ are given by $\mathbf{B}_{L,k}^{(i)} = \left[\mathbf{A}_{L,k}^{(i)} \,||\, \mathbf{F}_{L,k}^{(i)}\right]$.

(d) Next, it generates the components for each level. For each level $\mathsf{level} \in [1, L]$, do the following:

i. Choose matrices $\mathbf{S}_{\mathsf{level}}^{(0)}, \mathbf{S}_{\mathsf{level}}^{(1)} \leftarrow \chi^{n \times n}$ and $\mathbf{E}_{\mathsf{level}}^{(i,0)}, \mathbf{E}_{\mathsf{level}}^{(i,1)} \leftarrow \chi^{5n \times m}$ for $i \leq \ell_{\mathrm{PRG}}$. If either $\mathbf{S}_{\mathsf{level}}^{(0)}$ or $\mathbf{S}_{\mathsf{level}}^{(1)}$ has determinant zero, then set it to be $\mathbf{I}_n$.

ii. For $b \in \{0,1\}$, set matrix $\mathbf{D}_{\mathsf{level}}^{(i,b)}$ as a permutation of the matrix blocks of $\mathbf{B}_{\mathsf{level}}^{(i)}$ according to the permutation $\sigma_{\mathsf{level},b}^{(i)}(\cdot)$. More formally, for $i \leq \ell_{\mathrm{PRG}}$, set

$$\mathbf{D}_{\mathsf{level}}^{(i,b)} = \begin{bmatrix} \mathbf{B}^{(i)}_{\mathsf{level}, \sigma_{\mathsf{level},b}^{(i)}(1)} \\ \vdots \\ \mathbf{B}^{(i)}_{\mathsf{level}, \sigma_{\mathsf{level},b}^{(i)}(5)} \end{bmatrix}.$$

iii. Set $\mathbf{M}_{\mathsf{level}}^{(i,b)} = \left(\mathbf{I}_5 \otimes \mathbf{S}_{\mathsf{level}}^{(b)}\right) \cdot \mathbf{D}_{\mathsf{level}}^{(i,b)} + \mathbf{E}_{\mathsf{level}}^{(i,b)}$ for $i \leq \ell_{\mathrm{PRG}}$.

iv. Compute $\mathbf{C}_{\mathsf{level}}^{(i,b)} \leftarrow \mathsf{SamplePre}(\mathbf{B}_{\mathsf{level}-1}^{(i)}, T_{\mathsf{level}-1}^{(i)}, \sigma, \mathbf{M}_{\mathsf{level}}^{(i,b)})$

(e) Output $\left(\left\{\mathbf{B}_{0,1}^{(i)}\right\}_i, \{(\mathbf{C}_{\mathsf{level}}^{(i,0)}, \mathbf{C}_{\mathsf{level}}^{(i,1)})\}_{i \leq \ell_{\mathrm{PRG}}, \mathsf{level} \leq L}\right)$.

Figure 1: Routine Comp-Gen

- $\mathbf{S}_j = \mathbf{S}_j^{(z_{\mathsf{inp}(j)})}, \quad \mathbf{E}_j^{(i)} = \mathbf{E}_j^{(i, z_{\mathsf{inp}(j)})}, \quad \mathbf{C}_j^{(i)} = \mathbf{C}_j^{(i, z_{\mathsf{inp}(j)})}$ for all $j \leq L$

- $\mathbf{\Gamma}_{j^*} = \prod_{j=1}^{j^*} \mathbf{S}_j$ for all $j^* \leq L$

```
Comp-Eval
Input: Input string $z$, Components $\left(\left\{\mathbf{B}_{0,1}^{(i)}\right\}_i, \{(\mathbf{C}_{\mathsf{level}}^{(i,0)}, \mathbf{C}_{\mathsf{level}}^{(i,1)})\}_{i\leq\ell_{\mathrm{PRG}},\mathsf{level}\leq L}\right)$.
Output: $y \in \{0,1,\perp\}$.
  (a) For each $i \in [1, \ell_{\mathrm{PRG}}]$, do the following
      i. Set $\mathbf{M}^{(i)} = \mathbf{B}_{0,1}^{(i)}$.
      ii. For $j = 1$ to $L$, do the following
          - If $z_{\mathsf{inp}(j)} = 0$, set $\mathbf{M}^{(i)} = \mathbf{M}^{(i)} \cdot \mathbf{C}_j^{(i,0)}$. Else, set $\mathbf{M}^{(i)} = \mathbf{M}^{(i)} \cdot \mathbf{C}_j^{(i,1)}$.
  (b) Compute $\mathbf{M} = \sum_i \mathbf{M}^{(i)}$ and do the following
      - If $\|\mathbf{M}\|_\infty \leq q^{1/8}$, output 0.
      - Otherwise, if $\|\mathbf{M}\|_\infty \leq q^{1/2}$, output 1.
      - Else output $\perp$.
```

Figure 2: Routine Comp-Eval

- $\mathbf{\Delta}_{j^*}^{(i)} = \mathbf{B}_{0,1}^{(i)} \cdot \left(\prod_{j=1}^{j^*} \mathbf{C}_j^{(i)}\right)$, $\quad \widetilde{\mathbf{\Delta}}_{j^*}^{(i)} = \mathbf{\Gamma}_{j^*} \cdot \mathbf{B}_{j^*,\mathsf{st}_{j^*}^{(i)}}^{(i)}$, $\quad \mathbf{Err}_{j^*}^{(i)} = \mathbf{\Delta}_{j^*}^{(i)} - \widetilde{\mathbf{\Delta}}_{j^*}^{(i)}$ for all $j^* \leq L$
- For any string $x \in \{0,1\}^{\ell_{\mathrm{PRG}}}$, $\mathbf{A}_x = \sum_{i:x_i=0} \mathbf{A}_{L,\mathsf{rej}^{(i)}}^{(i)} + \sum_{i:x_i=1} \mathbf{A}_{L,\mathsf{acc}^{(i)}}^{(i)}$
- Similarly, let $\mathbf{B}_x = \sum_{i:x_i=0} \mathbf{B}_{L,\mathsf{rej}^{(i)}}^{(i)} + \sum_{i:x_i=1} \mathbf{B}_{L,\mathsf{acc}^{(i)}}^{(i)}$ & $\mathbf{F}_x = \sum_{i:x_i=0} \mathbf{F}_{L,\mathsf{rej}^{(i)}}^{(i)} + \sum_{i:x_i=1} \mathbf{F}_{L,\mathsf{acc}^{(i)}}^{(i)}$ & $\mathbf{E}_x = \sum_{i:x_i=0} \mathbf{E}_{L,\mathsf{rej}^{(i)}}^{(i)} + \sum_{i:x_i=1} \mathbf{E}_{L,\mathsf{acc}^{(i)}}^{(i)}$.

Observe that the Comp-Eval algorithm computes matrix $\mathbf{M} = \sum_{i=1}^{\ell_{\mathrm{PRG}}} \mathbf{\Delta}_L^{(i)}$. First, we show that for all $i \leq \ell_{\mathrm{PRG}}$, $j^* \leq L$, $\mathbf{Err}_{j^*}^{(i)}$ is small and bounded. This would help us in arguing that matrices $\mathbf{M} = \sum_{i=1}^{\ell_{\mathrm{PRG}}} \mathbf{\Delta}_L^{(i)}$ and $\widetilde{\mathbf{M}} = \sum_{i=1}^{\ell_{\mathrm{PRG}}} \widetilde{\mathbf{\Delta}}_L^{(i)}$ are very close to each other. We then prove the below bounds on $\mathbf{M}$ by proving the corresponding bounds on $\widetilde{\mathbf{M}}$ in each of the cases.

$$\|\mathbf{M}\|_\infty \begin{cases} < q^{1/8} & \text{when } \mathsf{BP}(z) = \beta \text{ and } \mathsf{msg} = 0 \\ \in (q^{1/8}, q^{1/2}) & \text{when } \mathsf{BP}(z) = \beta \text{ and } \mathsf{msg} = 1 \\ > q^{1/2} & \text{when } \mathsf{BP}(z) \neq \beta \end{cases}$$

First, we show that $\mathbf{Err}_{j^*}^{(i)}$ is bounded with the help of the following claim.

Claim 4.1. ([GKW17a, Claim 4.1]) $\forall\, i \in \{1,\ldots,\ell_{\mathrm{PRG}}\}, j^* \in \{1,\ldots,L\}, \quad \left\|\mathbf{Err}_{j^*}^{(i)}\right\|_\infty \leq j^* \cdot (m^2 \cdot \sigma)^{j^*}$.

The remaining proof of the lemma will have two parts, (1) when $\mathsf{BP}(z) = \beta$ and (2) when $\mathsf{BP}(z) \neq \beta$. Recall that the Comp-Eval algorithm computes matrix $\mathbf{M} = \sum_{i=1}^{\ell_{\mathrm{PRG}}} \mathbf{\Delta}_L^{(i)}$. Let $\widetilde{\mathbf{M}} = \sum_{i=1}^{\ell_{\mathrm{PRG}}} \widetilde{\mathbf{\Delta}}_L^{(i)}$ and $\mathsf{Err} = \sum_{i=1}^{\ell_{\mathrm{PRG}}} \mathbf{Err}_L^{(i)}$. Also, we parse these matrices as $\mathbf{M} = \left[\mathbf{M}^{(1)} \,||\, \mathbf{M}^{(2)}\right]$, $\widetilde{\mathbf{M}} = \left[\widetilde{\mathbf{M}}^{(1)} \,||\, \widetilde{\mathbf{M}}^{(2)}\right]$ and $\mathsf{Err} = \left[\mathsf{Err}^{(1)} \,||\, \mathsf{Err}^{(2)}\right]$, where $\mathbf{M}^{(1)}, \widetilde{\mathbf{M}}^{(1)}$ and $\mathsf{Err}^{(1)}$ are $n \times n$ (square) matrices.

First, note that $\mathbf{M} = \widetilde{\mathbf{M}} + \mathsf{Err}$. Using Claim 4.1, we can write that

$$\|\mathsf{Err}\|_\infty = \left\| \sum_{i=1}^{\ell_{\mathrm{PRG}}} \left(\mathbf{\Delta}_L^{(i)} - \widetilde{\mathbf{\Delta}}_L^{(i)}\right) \right\|_\infty \leq \sum_{i=1}^{\ell_{\mathrm{PRG}}} \left\| \mathbf{\Delta}_L^{(i)} - \widetilde{\mathbf{\Delta}}_L^{(i)} \right\|_\infty \leq \ell_{\mathrm{PRG}} \cdot L \cdot \left(m^2 \cdot \sigma\right)^L = \mathsf{Bd}. \tag{1}$$

Next, consider the following scenarios.

Part 1: $\mathsf{BP}(z) = \beta$**.** First, recall that the top level matrices always satisfy the following constraints during honest obfuscation:

$$\sum_{i=1}^{\ell_{\mathrm{PRG}}} \mathbf{B}^{(i)}_{L,\mathsf{st}_L^{(i)}} = \mathbf{B}_\beta = [\mathbf{A}_\beta \,||\, \mathbf{A}_\beta \cdot \mathbf{S} + \mathbf{E}_\beta] = \begin{cases} \left[\mathbf{0}^{n\times n} \,||\, \mathbf{E}_\beta\right] & \text{if } \mathsf{msg} = 0 \\ \left[q^{1/4}\cdot \mathbf{I}_n \,||\, q^{1/4}\cdot \mathbf{S} + \mathbf{E}_\beta\right] & \text{if } \mathsf{msg} = 1 \end{cases}$$

Note that

$$\widetilde{\mathbf{M}} = \sum_{i=1}^{\ell_{\mathrm{PRG}}} \widetilde{\mathbf{\Delta}}_L^{(i)} = \sum_{i=1}^{\ell_{\mathrm{PRG}}} \mathbf{\Gamma}_L \cdot \mathbf{B}^{(i)}_{L,\mathsf{st}_L^{(i)}} = \mathbf{\Gamma}_L \cdot \sum_{i=1}^{\ell_{\mathrm{PRG}}} \mathbf{B}^{(i)}_{L,\mathsf{st}_L^{(i)}} = \begin{cases} \left[\mathbf{0}^{n\times n} \,||\, \mathbf{\Gamma}_L \cdot \mathbf{E}_\beta\right] & \text{if } \mathsf{msg} = 0 \\ \mathbf{\Gamma}_L \cdot \left[q^{1/4}\cdot \mathbf{I}_n \,||\, q^{1/4}\cdot \mathbf{S} + \mathbf{E}_\beta\right] & \text{if } \mathsf{msg} = 1. \end{cases}$$

Next, we consider the following two cases dependending upon the message being obfuscated — (1) $\mathsf{msg} = 0$, (2) $\mathsf{msg} = 1$.

Case 1 ($\mathsf{msg} = 0$)**.** In this case, we bound the the l_∞ norm of the output matrix $\mathbf{M}$ (computed during evaluation) by $q^{1/8}$. We do this by bounding the norm of $\widetilde{\mathbf{M}}$ and using the error bound in Equation 1. Recall that when $\mathsf{msg} = 0$, $\widetilde{\mathbf{M}} = \left[\mathbf{0}^{n\times n} \,||\, \mathbf{\Gamma}_L \cdot \mathbf{E}_\beta\right]$. First, we bound the norms of $\mathbf{\Gamma}_L$ and $\mathbf{E}_\beta$ as follows.

$$\begin{aligned} \|\mathbf{E}_\beta\|_\infty &= \left\| \sum_{i:\beta_i=0} \mathbf{E}^{(i)}_{L,\mathsf{rej}^{(i)}} + \sum_{i:\beta_i=1} \mathbf{E}^{(i)}_{L,\mathsf{acc}^{(i)}} \right\|_\infty \\ &\le \sum_{i:\beta_i=0} \left\|\mathbf{E}^{(i)}_{L,\mathsf{rej}^{(i)}}\right\|_\infty + \sum_{i:\beta_i=1} \left\|\mathbf{E}^{(i)}_{L,\mathsf{acc}^{(i)}}\right\|_\infty \le \ell_{\mathrm{PRG}}\cdot\sigma\cdot m^{3/2} < \ell_{\mathrm{PRG}}\cdot \sigma \cdot m^2. \end{aligned} \tag{2}$$

The last inequality follows from the fact that the matrices $\mathbf{E}^{(i)}_{L,\mathsf{acc}^{(i)}}, \mathbf{E}^{(i)}_{L,\mathsf{rej}^{(i)}}$ are sampled from truncated gaussian distribution. We can also write that,

$$\|\mathbf{\Gamma}_L\|_\infty = \left\| \prod_{j=1}^{L} \mathbf{S}_j \right\|_\infty \le \prod_{j=1}^{L} \|\mathbf{S}_j\|_\infty \le (\sigma \cdot n \cdot \sqrt{m})^L < (\sigma\cdot m^2)^L. \tag{3}$$

This implies,

$$\left\|\widetilde{\mathbf{M}}\right\|_\infty = \|\mathbf{\Gamma}_L \cdot \mathbf{E}_\beta\|_\infty \le \|\mathbf{\Gamma}_L\|_\infty \cdot \|\mathbf{E}_\beta\|_\infty < (\sigma\cdot m^2)^L \cdot \ell_{\mathrm{PRG}} \cdot \sigma \cdot m^2 = \ell_{\mathrm{PRG}} \cdot (\sigma \cdot m^2)^{L+1}.$$

Now we bound the l_∞ norm of $\mathbf{M}$. Recall that, $\|\mathsf{Err}\|_\infty \le \ell_{\mathrm{PRG}} \cdot L \cdot (\sigma\cdot m^2)^L$. Therefore,

$$\begin{aligned} \|\mathbf{M}\|_\infty = \left\|\widetilde{\mathbf{M}} + \mathsf{Err}\right\|_\infty \le \left\|\widetilde{\mathbf{M}}\right\|_\infty + \|\mathsf{Err}\|_\infty &< \ell_{\mathrm{PRG}} \cdot L \cdot (\sigma\cdot m^2)^{L+1} + \ell_{\mathrm{PRG}} \cdot L \cdot (\sigma \cdot m^2)^L \\ &< \ell_{\mathrm{PRG}} \cdot (L+1) \cdot (\sigma\cdot m^2)^{L+1} < q^{1/8}. \end{aligned}$$

The last inequality follows from the constraints described in the construction. Thus, matrix $\mathbf{M}$ (computed during evaluation) always satisfies the condition that $\|\mathbf{M}\|_\infty < q^{1/8}$ if $\mathsf{msg} = 0$.

Case 2 ($\mathsf{msg} = 1$)**.** In this case, we prove that the l_∞ norm of the output matrix $\mathbf{M}$ (computed during evaluation) lies in $(q^{1/8}, q^{1/2})$. We do this by first computing upper and lower bounds on $\left\|\widetilde{\mathbf{M}}\right\|_\infty$ and using the bound on Err from Equation 1. Recall that when $\mathsf{msg} = 1$, $\widetilde{\mathbf{M}} = \left[q^{1/4}\cdot\mathbf{\Gamma}_L \,||\, q^{1/4}\cdot \mathbf{\Gamma}_L \cdot \mathbf{S} + \mathbf{\Gamma}_L \cdot \mathbf{E}_\beta\right]$. To prove a bound on $\left\|\widetilde{\mathbf{M}}\right\|_\infty$, we first prove bounds on individual components of $\widetilde{\mathbf{M}} : \mathbf{\Gamma}_L, \mathbf{S}, \mathbf{E}_\beta$.

By Equation 3, we have $\|\mathbf{\Gamma}_L\|_\infty < (\sigma\cdot m^2)^L$. Note that during obfuscation we sample secret matrices $\mathbf{S}^{(b)}_{\mathsf{level}}$ (for each level and bit b) such that they are short and *always* invertible. Therefore, matrix $\mathbf{\Gamma}_L$ (which is product of L secret matrices) is also invertible. Thus, we can write that $\|\mathbf{\Gamma}_L\|_\infty \geq 1$. The lower bound of 1 follows from the fact that $\mathbf{\Gamma}_L$ is non-singular (and integral) matrix. By Equation 2, we know that $\|\mathbf{E}_\beta\|_\infty < \ell_{\mathrm{PRG}} \cdot \sigma \cdot m^2$. Also, $\|\mathbf{S}\|_\infty \leq \sigma \cdot n \cdot \sqrt{m} < \sigma \cdot m^2$ as $\mathbf{S}$ is sampled from truncated gaussian distribution.

We finally prove bounds on $\left\|\widetilde{\mathbf{M}}\right\|_\infty$. We know that $\widetilde{\mathbf{M}}^{(1)} = q^{1/4} \cdot \mathbf{\Gamma}_L$ and $\widetilde{\mathbf{M}}^{(2)} = q^{1/4} \cdot \mathbf{\Gamma}_L \cdot \mathbf{S} + \mathbf{\Gamma}_L \cdot \mathbf{E}_\beta$.

$$\left\|\widetilde{\mathbf{M}}\right\|_\infty \geq \left\|\widetilde{\mathbf{M}}^{(1)}\right\|_\infty = q^{1/4} \cdot \|\mathbf{\Gamma}_L\|_\infty \geq q^{1/4}$$

$$\left\|\widetilde{\mathbf{M}}^{(1)}\right\|_\infty \leq q^{1/4} \cdot \|\mathbf{\Gamma}_L\|_\infty < q^{1/4} \cdot (\sigma \cdot m^2)^L$$

$$\begin{aligned}\left\|\widetilde{\mathbf{M}}^{(2)}\right\|_\infty \leq q^{1/4} \cdot \|\mathbf{\Gamma}_L\|_\infty \cdot \|\mathbf{S}\|_\infty + \|\mathbf{\Gamma}_L\|_\infty \cdot \|\mathbf{E}_\beta\|_\infty &< q^{1/4} \cdot (\sigma \cdot m^2)^{L+1} + \ell_{\mathrm{PRG}} \cdot (\sigma \cdot m^2)^{L+1} \\ &< q^{1/4} \cdot (\ell_{\mathrm{PRG}} + 1) \cdot (\sigma \cdot m^2)^{L+1}\end{aligned}$$

This implies,

$$\begin{aligned}\left\|\widetilde{\mathbf{M}}\right\|_\infty \leq \left\|\widetilde{\mathbf{M}}^{(1)}\right\|_\infty + \left\|\widetilde{\mathbf{M}}^{(2)}\right\|_\infty &< q^{1/4} \cdot (\sigma \cdot m^2)^L + q^{1/4} \cdot (\ell_{\mathrm{PRG}} + 1) \cdot (\sigma \cdot m^2)^{L+1} \\ &< q^{1/4} \cdot (\ell_{\mathrm{PRG}} + 2) \cdot (\sigma \cdot m^2)^{L+1} < q^{1/4} \cdot q^{1/8} < q^{3/8}\end{aligned}$$

The last inequality follows from the constraints described in the construction. Next, we show that matrix $\mathbf{M}^{(1)}$ has large entries. In other words, matrix $\mathbf{M}$ has high l_∞ norm. Concretely,

$$\|\mathbf{M}\|_\infty = \left\|\widetilde{\mathbf{M}} + \mathsf{Err}\right\|_\infty \leq \left\|\widetilde{\mathbf{M}}\right\|_\infty + \|\mathsf{Err}\|_\infty = q^{3/8} + \mathsf{Bd} < q^{3/8} + q^{1/8} < q^{1/2}.$$

$$\|\mathbf{M}\|_\infty = \left\|\widetilde{\mathbf{M}} + \mathsf{Err}\right\|_\infty \geq \left\|\widetilde{\mathbf{M}}\right\|_\infty - \|\mathsf{Err}\|_\infty \geq \left\|\widetilde{\mathbf{M}}^{(1)}\right\|_\infty - \|\mathsf{Err}\|_\infty \geq q^{1/4} - \mathsf{Bd} > q^{1/4} - q^{1/8} > q^{1/8}.$$

Therefore, if $\mathsf{msg} = 1$, $\|\mathbf{M}\|_\infty \in (q^{1/8}, q^{1/2})$ and the evaluation always outputs 1.

Part 2: $\mathsf{BP}(z) \neq \beta$**.** In this case, we prove that the l_∞ norm of output matrix $\mathbf{M}$ is at least $q^{1/2}$. Let $x = \mathsf{BP}(z)$ and δ_x be the edit distance between x and β, which is clearly greater than 0 if $x \neq \beta$. By construction, $\widetilde{\mathbf{M}} = \mathbf{\Gamma}_L \cdot [\mathbf{A}_x \,||\, \mathbf{A}_x \cdot \mathbf{S} + \mathbf{E}_x + \delta_x \cdot \mathbf{D}]$ and $\mathbf{M} = \widetilde{\mathbf{M}} + \mathsf{Err}$. We now split this case into two subcases: 1) $\left\|\mathbf{M}^{(1)}\right\|_\infty > q^{1/2}$ and 2) $\left\|\mathbf{M}^{(1)}\right\|_\infty \leq q^{1/2}$.

Case 1. $\left\|\mathbf{M}^{(1)}\right\|_\infty > q^{1/2}$. In this case, $\|\mathbf{M}\|_\infty > q^{1/2}$ and the evaluator always outputs $\perp$.

Case 2. $\left\|\mathbf{M}^{(1)}\right\|_\infty \leq q^{1/2}$. In this case, we prove that $\mathbf{M}^{(2)}$ has high l_∞ norm. Recall that $\|\mathbf{S}\|_\infty \leq \sigma \cdot n \cdot \sqrt{m} < \sigma \cdot m^2$ as $\mathbf{S}$ is sampled from truncated gaussian distribution and $\|\mathbf{E}_x\|_\infty \leq \ell_{\mathrm{PRG}} \cdot \sigma \cdot m^2$ by an analysis similar to Equation 2. Also, $\|\mathbf{\Gamma}_L\|_\infty < (\sigma \cdot m^2)^L$ by Equation 3. We now prove an upper bound on norm of $\mathbf{\Gamma}_L \cdot [\mathbf{A}_x \cdot \mathbf{S} + \mathbf{E}_x]$.

$$\begin{aligned}\|\mathbf{\Gamma}_L \cdot \mathbf{A}_x\|_\infty &\leq \left\|\mathbf{M}^{(1)}\right\|_\infty + \left\|\mathsf{Err}^{(1)}\right\|_\infty \leq q^{1/2} + \mathsf{Bd} \\ \|\mathbf{\Gamma}_L \cdot \mathbf{A}_x \cdot \mathbf{S} + \mathbf{\Gamma}_L \cdot \mathbf{E}_x\|_\infty &\leq \|\mathbf{\Gamma}_L \cdot \mathbf{A}_x\|_\infty \cdot \|\mathbf{S}\|_\infty + \|\mathbf{\Gamma}_L\|_\infty \cdot \|\mathbf{E}_x\|_\infty \\ &\leq (q^{1/2} + \mathsf{Bd}) \cdot \sigma \cdot m^2 + \ell_{\mathrm{PRG}} \cdot (\sigma \cdot m^2)^{L+1} \\ &\leq q^{1/2} \cdot \sigma \cdot m^2 + \ell_{\mathrm{PRG}} \cdot L \cdot (\sigma \cdot m^2)^{L+1} + \ell_{\mathrm{PRG}} \cdot (\sigma \cdot m^2)^{L+1} \\ &< q^{1/2} \cdot \sigma \cdot m^2 + \ell_{\mathrm{PRG}} \cdot (L+1) \cdot (\sigma \cdot m^2)^{L+1} < q^{1/2} \cdot q^{1/8} + q^{1/8} < 1/2 \cdot q^{3/4}\end{aligned} \tag{4}$$

The last 2 inequalities follow from the constraints described in the construction. As $\mathbf{\Gamma}_L \cdot \mathbf{D} = \left[q^{3/4} \cdot \mathbf{\Gamma}_L \,||\, \mathbf{0}^{n \times (m-2\cdot n)} \right]$, we know that $\|\mathbf{\Gamma}_L \cdot \mathbf{D}\|_\infty = q^{3/4} \cdot \|\mathbf{\Gamma}_L\|_\infty$, which lies in $[q^{3/4}, q^{3/4} \cdot (\sigma \cdot m^2)^L]$ as discussed earlier. This along with Equation 4 implies the following upper bound on $\left\|\widetilde{\mathbf{M}}^{(2)}\right\|_\infty$.

$$\begin{aligned}
\left\|\widetilde{\mathbf{M}}^{(2)}\right\|_\infty &= \|\mathbf{\Gamma}_L \cdot [\mathbf{A}_x \cdot \mathbf{S} + \mathbf{E}_x + \delta_x \cdot \mathbf{D}]\|_\infty \\
&\leq \|\mathbf{\Gamma}_L \cdot \mathbf{A}_x \cdot \mathbf{S} + \mathbf{\Gamma}_L \cdot \mathbf{E}_x\|_\infty + \delta_x \cdot \|\mathbf{\Gamma}_L \cdot \mathbf{D}\|_\infty \\
&< 1/2 \cdot q^{3/4} + \ell_{\mathrm{PRG}} \cdot \|\mathbf{\Gamma}_L \cdot \mathbf{D}\|_\infty \leq 1/2 \cdot q^{3/4} + q^{3/4} \cdot \ell_{\mathrm{PRG}} \cdot (\sigma \cdot m^2)^L < q^{3/4} \cdot q^{1/8} = q^{7/8}
\end{aligned}$$

The last inequality follows from the constraints described in the construction. We can also prove the following lower bound on $\left\|\widetilde{\mathbf{M}}^{(2)}\right\|_\infty$.

$$\begin{aligned}
\left\|\widetilde{\mathbf{M}}^{(2)}\right\|_\infty &= \|\mathbf{\Gamma}_L \cdot [\mathbf{A}_x \cdot \mathbf{S} + \mathbf{E}_x + \delta_x \cdot \mathbf{D}]\|_\infty \\
&\geq - \|\mathbf{\Gamma}_L \cdot \mathbf{A}_x \cdot \mathbf{S} + \mathbf{\Gamma}_L \cdot \mathbf{E}_x\|_\infty + \|\mathbf{\Gamma}_L \cdot \mathbf{D}\|_\infty > -1/2 \cdot q^{3/4} + q^{3/4} = 1/2 \cdot q^{3/4}
\end{aligned}$$

Now, we prove upper and lower bounds on $\mathbf{M}^{(2)} = \widetilde{\mathbf{M}}^{(2)} + \mathsf{Err}^{(2)}$.

$$q^{1/2} < 1/2 \cdot q^{3/4} - q^{1/8} < 1/2 \cdot q^{3/4} - \mathsf{Bd} \leq \left\|\mathbf{M}^{(2)}\right\|_\infty \leq q^{7/8} + \mathsf{Bd} < q^{7/8} + q^{1/8} < q/2$$

This implies, $\left\|\mathbf{M}^{(2)}\right\|_\infty > q^{1/2}$ in this case. Therefore, $\|\mathbf{M}\|_\infty > q^{1/2}$ and the evaluator always outputs $\perp$. ■

Using the above lemma, we can now argue the correctness of our scheme. First, we need to show correctness for the case when $P(x) = \alpha$.

Claim 4.2. For any security parameter $\lambda \in \mathbb{N}$, any input $x \in \{0,1\}^{\ell_{\mathrm{in}}}$, any program $P \in \mathcal{C}_{\ell_{\mathrm{in}}, \ell_{\mathrm{out}}, d}$ and any message $\mathsf{msg} \in \{0,1\}$, if $P(x) = \alpha$, then

$$\mathsf{Eval}(\mathsf{Obf}(1^\lambda, P, \mathsf{msg}, \alpha), x) = \mathsf{msg}.$$

Proof. First, the obfuscator encrypts the program P using an LHE secret key $\mathsf{lhe.sk}$, and sets $\mathsf{ct} \leftarrow \mathsf{LHE.Enc}(\mathsf{lhe.sk}, P)$. The evaluator evaluates the LHE ciphertext on universal circuit $U_x(\cdot)$, which results in an evaluated ciphertext $\widetilde{\mathbf{ct}}$. Now, by the correctness of the LHE scheme, decryption of $\widetilde{\mathbf{ct}}$ using $\mathsf{lhe.sk}$ outputs α. Therefore, $\mathrm{PRG}.\mathsf{Eval}(\mathsf{pp}, \mathsf{LHE.Dec}(\mathsf{lhe.sk}, \widetilde{\mathbf{ct}})) = \beta$, where $\mathsf{pp} \leftarrow \mathrm{PRG}.\mathsf{Setup}(1^\lambda)$.[10] Then, using Lemma 4.1, we can argue that $\mathsf{Comp\text{-}Eval}$ outputs msg, and thus Eval outputs msg. ■

Claim 4.3. For all security parameters λ, inputs $x \in \{0,1\}^{\ell_{\mathrm{in}}}$, programs $P \in \mathcal{C}_{\ell_{\mathrm{in}}, \ell_{\mathrm{out}}, d}$, $\alpha \in \{0,1\}^{\ell_{\mathrm{out}}}$ such that $P(x) \neq \alpha$ and $\mathsf{msg} \in \{0,1\}$,

$$\mathsf{Eval}(\mathsf{Obf}(1^\lambda, P, \mathsf{msg}, \alpha), x) = \perp$$

Proof. Fix any security parameter λ, program P, α, x such that $P(x) \neq \alpha$ and message msg. The evaluator evaluates the LHE ciphertext on universal circuit $U_x(\cdot)$, which results in an evaluated ciphertext $\widetilde{\mathbf{ct}}$. Now, by the correctness of the LHE scheme, decryption of $\widetilde{\mathbf{ct}}$ using $\mathsf{lhe.sk}$ does not output α. Therefore, by the perfect injectivity of PRG scheme, for all $\mathsf{pp} \leftarrow \mathrm{PRG}.\mathsf{Setup}(1^\lambda)$, we have $\mathrm{PRG}.\mathsf{Eval}(\mathsf{pp}, \mathsf{LHE.Dec}(\mathsf{lhe.sk}, \widetilde{\mathbf{ct}})) \neq \beta$. Then, using Lemma 4.1, we can argue that $\mathsf{Comp\text{-}Eval}$ outputs $\perp$, and thus Eval outputs $\perp$. ■

[10] As before, we are overloading the notation and using $\mathsf{LHE.Dec}$ to decrypt multiple ciphertexts.

4.3 Security

In this subsection, we prove the security of the above construction. Concretely, we prove the following theorem.

Theorem 4.1. Assuming that LHE is a secure leveled homomorphic encryption scheme, and PRG is a secure perfectly injective pseudorandom generator, lattice trapdoors are secure and $(n, 2n \cdot \ell_{\text{PRG}}, m-n, q, \chi)$-LWE-ss, $(n, 5m \cdot \ell_{\text{PRG}}, n, q, \chi)$-LWE-ss assumptions hold, the lockable obfuscation construction described in Section 4.1 is secure as per Definition 2.2.

Proof. We prove the above theorem by proving that our construction is computationally indistinguishable from the construction provided in [GKW17a, Appendix D] that uses perfectly injective PRGs. Note that Goyal et al. [GKW17a] construct a simulator $\mathsf{Sim}(1^\lambda, 1^{|P|}, 1^{|\alpha|})$ and prove that their construction is computationally indstinguishable from the simulator. By a standard hybrid argument, this implies that our construction is computationally indstinguishable from the simulator. Formally, we prove the following theorem.

Theorem 4.2. Assuming that PRG is a secure perfectly injective pseudorandom generator and $(n, 2n \cdot \ell_{\text{PRG}}, m-n, q, \chi)$-LWE-ss assumption holds, the lockable obfuscation construction described in Section 4.1 is computationally indistinguishable[11] from [GKW17a, Appendix D] construction that uses perfectly injective PRGs.

We prove the theorem using the following sequence of hybrids. The first hybrid corresponds to the security game in which the challenger uses our lockable obfuscation scheme (Section 4.1) for obfuscating the challenge program. The last hybrid corresponds to the security game in which the challenger uses lockable obfuscation scheme provided in [GKW17a]. We note that some portions of the proof are similar to those used in [GKW17a].

Game **0**. This game correponds to the challenger using our lockable obfuscation scheme for obfuscating the challenge program.

1. The adversary sends a program P and message msg to the challenger.
2. The challenger first chooses the LWE parameters n, m, q, σ, χ and ℓ_{PRG}. Recall L denotes the length of the branching programs.
3. The challenger then chooses $(\mathsf{sk}, \mathsf{ek}) \leftarrow \mathsf{LHE.Setup}(1^\lambda, 1^{d \log d})$ and sets $\mathsf{ct} \leftarrow \mathsf{LHE.Enc}(\mathsf{sk}, P)$.
4. Next, it chooses a uniformly random string $\alpha \leftarrow \{0,1\}^{\ell_{\text{out}}}$, runs $\mathsf{pp} \leftarrow \text{PRG}.\mathsf{Setup}(1^\lambda)$ and sets $\beta = \text{PRG}.\mathsf{Eval}(\mathsf{pp}, \alpha)$.
5. Next, consider the following program Q. It takes as input an LHE ciphertext ct, has sk hardwired and does the following: it decrypts the input ciphertext ct to get string x and outputs $\text{PRG}.\mathsf{Eval}(\mathsf{pp}, x)$. For $i \le \ell_{\text{PRG}}(\lambda)$, let $\mathsf{BP}^{(i)}$ denote the branching program that outputs the i^{th} bit of $\text{PRG}.\mathsf{Eval}(\mathsf{pp}, x)$.
6. For $i = 1$ to ℓ_{PRG} and $j = 0$ to $L-1$, it chooses $(\mathbf{B}_j^{(i)}, T_j^{(i)}) \leftarrow \mathsf{TrapGen}(1^{5n}, 1^m, q)$.
7. Let $\mathbf{D} = q^{3/4} \cdot \left[\mathbf{I}_n \,||\, \mathbf{0}^{n \times (m-2 \cdot n)}\right]$.
   (a) For the top level, it first chooses the matrices $\mathbf{A}_{L,k}^{(i)}$ (of dimension $n \times n$) for each $i \le \ell_{\text{PRG}}, k \le 5$, uniformly at random, subject to the following constraints:

$$\sum_{i:\beta_i=0} \mathbf{A}_{L,\mathsf{rej}^{(i)}}^{(i)} + \sum_{i:\beta_i=1} \mathbf{A}_{L,\mathsf{acc}^{(i)}}^{(i)} = \mathbf{0}^{n \times n} \text{ if } \mathsf{msg} = 0.$$

$$\sum_{i:\beta_i=0} \mathbf{A}_{L,\mathsf{rej}^{(i)}}^{(i)} + \sum_{i:\beta_i=1} \mathbf{A}_{L,\mathsf{acc}^{(i)}}^{(i)} = q^{1/4} \cdot \mathbf{I}_n \text{ if } \mathsf{msg} = 1.$$

[11] Consider a game in which the adversary sends a program P and message msg to the challenger, which either obfuscates (P, msg) using [GKW17a] construction or our construction and sends back the obfuscated program. No PPT adversary can distinguish the two scenarios with non-negligible advantage.

(b) It then samples a matrix $\mathbf{S} \leftarrow \chi^{n \times (m-n)}$, and matrices $\mathbf{E}^{(i)}_{L,\mathsf{rej}^{(i)}} \leftarrow \chi^{n \times (m-n)}, \mathbf{E}^{(i)}_{L,\mathsf{acc}^{(i)}} \leftarrow \chi^{n \times (m-n)}$ for each $i \leq \ell_{\mathrm{PRG}}$. Next, it chooses matrices $\mathbf{F}^{(i)}_{L,k}$ as follows

$$\begin{aligned}\mathbf{F}^{(i)}_{L,\mathsf{acc}^{(i)}} &= \mathbf{A}^{(i)}_{L,\mathsf{acc}^{(i)}} \cdot \mathbf{S} + \mathbf{E}^{(i)}_{L,\mathsf{acc}^{(i)}} + (1 - \beta_i) \cdot \mathbf{D} \\ \mathbf{F}^{(i)}_{L,\mathsf{rej}^{(i)}} &= \mathbf{A}^{(i)}_{L,\mathsf{rej}^{(i)}} \cdot \mathbf{S} + \mathbf{E}^{(i)}_{L,\mathsf{rej}^{(i)}} + \beta_i \cdot \mathbf{D} \\ \mathbf{F}^{(i)}_{L,k} &\leftarrow \mathbb{Z}_q^{n \times (m-n)} \text{ if } k \notin \{\mathsf{acc}^{(i)}, \mathsf{rej}^{(i)}\}\end{aligned}$$

(c) The top level matrices $\mathbf{B}^{(i)}_{L,k}$ for each $i \leq \ell_{\mathrm{PRG}}, k \leq 5$ are set to $\mathbf{B}^{(i)}_{L,k} = \left[\mathbf{A}^{(i)}_{L,k} \,||\, \mathbf{F}^{(i)}_{L,k}\right]$.

8. Next, it generates the components for each level. For each $i \in [1, \ell_{\mathrm{PRG}}]$ and each level $\mathsf{level} \in [1, L]$, do the following:
   (a) Choose matrices $\mathbf{S}^{(0)}_{\mathsf{level}}, \mathbf{S}^{(1)}_{\mathsf{level}} \leftarrow \chi^{n \times n}$ and $\mathbf{E}^{(i,0)}_{\mathsf{level}}, \mathbf{E}^{(i,1)}_{\mathsf{level}} \leftarrow \chi^{5n \times m}$ for $i \leq \ell_{\mathrm{PRG}}$. If either $\mathbf{S}^{(0)}_{\mathsf{level}}$ or $\mathbf{S}^{(1)}_{\mathsf{level}}$ has determinant zero, then set it to be $\mathbf{I}_n$.
   (b) For $b \in \{0, 1\}$, set matrix $\mathbf{D}^{(i,b)}_{\mathsf{level}}$ as a permutation of the matrix blocks of $\mathbf{B}^{(i)}_{\mathsf{level}}$ according to the permutation $\sigma^{(i)}_{\mathsf{level},b}(\cdot)$.
   (c) Set $\mathbf{M}^{(i,b)}_{\mathsf{level}} = \left(\mathbf{I}_5 \otimes \mathbf{S}^{(b)}_{\mathsf{level}}\right) \cdot \mathbf{D}^{(i,b)}_{\mathsf{level}} + \mathbf{E}^{(i,b)}_{\mathsf{level}}$ for $i \leq \ell_{\mathrm{PRG}}$.
   (d) Compute $\mathbf{C}^{(i,b)}_{\mathsf{level}} \leftarrow \mathsf{SamplePre}(\mathbf{B}^{(i)}_{\mathsf{level}-1}, T^{(i)}_{\mathsf{level}-1}, \sigma, \mathbf{M}^{(i,b)}_{\mathsf{level}})$
9. The challenger sends the final obfuscated program which consists of the LHE evaluation key ek, LHE encryption ct, together with the components $\left(\left\{\mathbf{B}^{(i)}_{0,1}\right\}_i, \left\{(\mathbf{C}^{(i,0)}_j, \mathbf{C}^{(i,1)}_j)\right\}_{i,j}\right)$ to the adversary.
10. The adversary outputs a bit b'.

Game 1: In this hybrid, the string β is chosen uniformly at random.

4. Next, it chooses a uniformly random string $\beta \leftarrow \{0,1\}^{\ell_{\mathrm{PRG}}}$.

Game 2: In this hybrid, the matrices $\mathbf{A}^{(i)}_{L,k}$ are chosen uniformly at random without any constraints.

7. (a) For the top level, it first chooses the matrices $\mathbf{A}^{(i)}_{L,k}$ (of dimension $n \times n$) for each $i \leq \ell_{\mathrm{PRG}}, k \leq 5$, uniformly at random without any constraints.

Game 3: In this hybrid, all the matrices $\mathbf{F}^{(i)}_{L,k}$ are chosen uniformly at random.

7. (b) It then samples matrices $\mathbf{R}^{(i)}_{L,\mathsf{rej}^{(i)}} \leftarrow \mathbb{Z}_q^{n \times (m-n)}, \mathbf{R}^{(i)}_{L,\mathsf{acc}^{(i)}} \leftarrow \mathbb{Z}_q^{n \times (m-n)}$ for each $i \leq \ell_{\mathrm{PRG}}$. Next, it chooses matrices $\mathbf{F}^{(i)}_{L,k}$ as follows.

$$\begin{aligned}\mathbf{F}^{(i)}_{L,\mathsf{acc}^{(i)}} &= \mathbf{R}^{(i)}_{L,\mathsf{acc}^{(i)}} + (1 - \beta_i) \cdot \mathbf{D} \\ \mathbf{F}^{(i)}_{L,\mathsf{rej}^{(i)}} &= \mathbf{R}^{(i)}_{L,\mathsf{rej}^{(i)}} + \beta_i \cdot \mathbf{D} \\ \mathbf{F}^{(i)}_{L,k} &\leftarrow \mathbb{Z}_q^{n \times (m-n)} \text{ if } k \notin \{\mathsf{acc}^{(i)}, \mathsf{rej}^{(i)}\}\end{aligned}$$

Game 4: In this hybrid, all the top level matrices $\mathbf{B}^{(i)}_{L,k}$ are chosen uniformly at random.

7. For the top level, for each $i \leq \ell_{\mathrm{PRG}}$ and $k \leq 5$, it chooses the matrices $\mathbf{B}^{(i)}_{L,k}$ uniformly at random from $\mathbb{Z}_q^{n \times m}$.

Game 5: In this hybrid, the top level matrices $\mathbf{B}^{(i)}_{L,k}$ are chosen according to GKW17 construction.

7. For the top level, for each $i \leq \ell_{\text{PRG}}$ and $k \leq 5$, it chooses the matrices $\mathbf{B}^{(i)}_{L,k}$ uniformly at random from $\mathbb{Z}_q^{n \times m}$ subject to the following constraints.

$$\sum_{i\,:\,\beta_i=0} \mathbf{B}^{(i)}_{L,\mathsf{rej}^{(i)}} + \sum_{i\,:\,\beta_i=1} \mathbf{B}^{(i)}_{L,\mathsf{acc}^{(i)}} = \begin{cases} \mathbf{0} & \text{if } \mathsf{msg} = 0. \\ \sqrt{q} \cdot \left[\mathbf{I}_n \,||\, \mathbf{0}^{n \times (m-n)}\right] & \text{if } \mathsf{msg} = 1. \end{cases}$$

Game 6: This hybrid corresponds to challenger using GKW17 lockable obfuscation scheme for obfuscating the challenge program.

4. Next, it chooses a uniformly random string $\alpha \leftarrow \{0,1\}^{\ell_{\text{out}}}$, runs $\mathsf{pp} \leftarrow \mathsf{PRG.Setup}(1^\lambda)$ and sets $\beta = \mathsf{PRG.Eval}(\mathsf{pp}, \alpha)$.

We now establish that Game 0 is indistinguishable from Game 6 using the following sequence of claims. For any adversary $\mathcal{A}$, let $p_i^{\mathcal{A}}$ denote the probability that the adversary outputs 1 in Game i.

Lemma 4.2. Assuming the security of PRG, for any PPT adversary A, there exists a negligible function $\text{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, we have $|p_0^{\mathcal{A}} - p_1^{\mathcal{A}}| \leq \text{negl}(\lambda)$.

Proof. Suppose there exists a PPT adverary $\mathcal{A}$ and a non-negligible function $\delta(\cdot)$ such that $|p_0^{\mathcal{A}} - p_1^{\mathcal{A}}| > \delta(\lambda)$ for all $\lambda \in \mathbb{N}$. We build a PPT algorithm $\mathcal{B}$ that uses $\mathcal{A}$ and breaks PRG security.

The PRG challenger $\mathcal{C}$ first samples PRG public parameters $\mathsf{pp} \leftarrow \mathsf{PRG.Setup}(1^\lambda)$ and a uniformly random bit $b \leftarrow \{0,1\}$. If $b = 0$, it samples $x \leftarrow \{0,1\}^{\ell_{\text{out}}}$ and evaluates $y = \mathsf{PRG.Eval}(\mathsf{pp}, x)$. Otherwise, it samples $y \leftarrow \{0,1\}^{\ell_{\text{PRG}}}$. $\mathcal{C}$ sends public parameters pp and challenge y to $\mathcal{B}$. $\mathcal{B}$ then receives a program P and a message msg from adversary $\mathcal{A}$. $\mathcal{B}$ obfuscates the program P and message msg using $\beta = y$, and sends the obfuscated program to $\mathcal{A}$. The adversary outputs a bit b', which $\mathcal{B}$ outptus as its guess to PRG challenger.

Note that $\mathcal{B}$ simulates Game 0 to $\mathcal{A}$ if $b = 0$, and simulates Game 1 to $\mathcal{A}$ if $b = 1$. Therefore, the advantage of $\mathcal{B}$ in PRG security game is non-negligible. ■

Lemma 4.3. For any adversary $\mathcal{A}$, there exists a negligible function $\text{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, we have $|p_1^{\mathcal{A}} - p_2^{\mathcal{A}}| \leq \text{negl}(\lambda)$.

Proof. This step is information theoretic, and uses the Leftover Hash Lemma (Corollary 2.1). Note that the difference between the two games is the way the matrices $\mathbf{A}^{(i)}_{L,k}$ are sampled. For each $i \leq \ell_{\text{PRG}}$, let $\mathsf{st}^{(i)} = \mathsf{acc}^{(i)}$ if $\beta_i = 1$ and $\mathsf{st}^{(i)} = \mathsf{rej}^{(i)}$ if $\beta_i = 0$. In both games, the matrices $\mathbf{A}^{(i)}_{L,k}$, for all (i,k) such that $(i,k) \neq (\ell_{\text{PRG}}, \mathsf{st}^{(\ell_{\text{PRG}})})$, are chosen unformly at random. In Game 2, the matrix $\mathbf{A}^{(\ell_{\text{PRG}})}_{L,\mathsf{st}^{(\ell_{\text{PRG}})}}$ is also chosen uniformly at random. In Game 1, the matrix $\mathbf{A}^{(\ell_{\text{PRG}})}_{L,\mathsf{st}^{(\ell_{\text{PRG}})}}$ is chosen as

$$\mathbf{A}^{(\ell_{\text{PRG}})}_{L,\mathsf{st}^{(\ell_{\text{PRG}})}} = \begin{cases} -\left(\sum_{i<\ell_{\text{PRG}}:\beta_i=0} \mathbf{A}^{(i)}_{L,\mathsf{rej}^{(i)}} + \sum_{i<\ell_{\text{PRG}}:\beta_i=1} \mathbf{A}^{(i)}_{L,\mathsf{acc}^{(i)}}\right) & \text{if } \mathsf{msg} = 0 \\ q^{1/4} \cdot \mathbf{I}_n - \left(\sum_{i<\ell_{\text{PRG}}:\beta_i=0} \mathbf{A}^{(i)}_{L,\mathsf{rej}^{(i)}} + \sum_{i<\ell_{\text{PRG}}:\beta_i=1} \mathbf{A}^{(i)}_{L,\mathsf{acc}^{(i)}}\right) & \text{if } \mathsf{msg} = 1 \end{cases}.$$

This can also be written as

$$\mathbf{A}^{(\ell_{\text{PRG}})}_{L,\mathsf{st}^{(\ell_{\text{PRG}})}} = q^{1/4} \cdot \mathsf{msg} \cdot \mathbf{I}_n - \mathbf{H} \cdot \mathbf{R},$$

where $\mathbf{H} = \left[\mathbf{A}^{(1)}_{L,\mathsf{rej}^{(1)}} \,||\, \mathbf{A}^{(1)}_{L,\mathsf{acc}^{(1)}} \,||\, \mathbf{A}^{(2)}_{L,\mathsf{rej}^{(2)}} \,||\, \cdots \,||\, \mathbf{A}^{(\ell_{\text{PRG}}-1)}_{L,\mathsf{acc}^{(\ell_{\text{PRG}}-1)}}\right]$ and $\mathbf{R} = \mathbf{u} \otimes \mathbf{I}_n \in \mathbb{Z}_q^{2n(\ell_{\text{PRG}}-1) \times n}$. Here $\mathbf{u} = (u_1, \ldots, u_{2 \cdot \ell_{\text{PRG}}-2})^T \in \{0,1\}^{2\ell_{\text{PRG}}-2}$ where $u_{2i} = \beta_i$ and $u_{2i-1} = 1 - \beta_i$ for all $i \leq \ell_{\text{PRG}} - 1$. That is, matrix $\mathbf{R}$ consists of $2\ell_{\text{PRG}} - 2$ submatrices where if $\beta_i = 1$, then its $2i^{th}$ submatrix is identity and $(2i-1)^{th}$

submatrix is zero, otherwise it is the opposite. Let $\mathcal{R}$ denote the distribution of matrix $\mathbf{R}$ as described above with β drawn uniformly from $\{0,1\}^{\ell_{\mathrm{PRG}}}$. Note that $\mathbf{H}_\infty(\mathcal{R}) = \ell_{\mathrm{PRG}} - 1$ (min-entropy of $\mathcal{R}$), and $\ell_{\mathrm{PRG}} > n^2 \cdot \log_2 q + \omega(\log n)$. Therefore, it follows (from Corollary 2.1) that

$$\left\{ \left(\mathbf{H}, \mathbf{A}^{(\ell_{\mathrm{PRG}})}_{L,\mathsf{st}^{(\ell_{\mathrm{PRG}})}} = q^{1/4} \cdot \mathsf{msg} \cdot \mathbf{I}_n - \mathbf{H} \cdot \mathbf{R} \right) \ : \ \mathbf{H} \leftarrow \mathbb{Z}_q^{n \times 2n(\ell_{\mathrm{PRG}}-1)}, \mathbf{R} \leftarrow \mathcal{R} \right\}$$

$$\approx_s$$

$$\left\{ \left(\mathbf{H}, \mathbf{A}^{(\ell_{\mathrm{PRG}})}_{L,\mathsf{st}^{(\ell_{\mathrm{PRG}})}} \right) \ : \ \mathbf{H} \leftarrow \mathbb{Z}_q^{n \times 2n(\ell_{\mathrm{PRG}}-1)}, \mathbf{A}^{(\ell_{\mathrm{PRG}})}_{L,\mathsf{st}^{(\ell_{\mathrm{PRG}})}} \leftarrow \mathbb{Z}_q^{n \times n} \right\}$$

Thus, $|p_1^{\mathcal{A}} - p_2^{\mathcal{A}}|$ is negligible in the security parameter for all adversaries $\mathcal{A}$. ∎

Lemma 4.4. Assuming $(n, 2n \cdot \ell_{\mathrm{PRG}}, q, \chi) - \mathsf{LWE\text{-}ss}$ is secure, for any PPT adversary $\mathcal{A}$, there exists a negligible function negl$(\cdot)$ such that for all $\lambda \in \mathbb{N}$, we have $|p_2^{\mathcal{A}} - p_3^{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ and a non-negligible function $\delta(\cdot)$ such that $|p_2^{\mathcal{A}} - p_3^{\mathcal{A}}| > \delta(\lambda)$ for all $\lambda \in \mathbb{N}$. We build a PPT adversary $\mathcal{B}$ that uses $\mathcal{A}$ and breaks LWE with short secrets assumption. The algorithm $\mathcal{B}$ proceeds as follows.

LWE-ss challenger $\mathcal{C}$ first samples a matrix $\mathbf{H} \leftarrow \mathbb{Z}_q^{2n \cdot \ell_{\mathrm{PRG}} \times n}$ and a bit $b \leftarrow \{0,1\}$. If $b = 0$, it samples $\mathbf{S} \leftarrow \chi^{n \times (m-n)}$, $\mathbf{E} \leftarrow \chi^{2n \cdot \ell_{\mathrm{PRG}} \times (m-n)}$ and sets $\mathbf{G} = \mathbf{H} \cdot \mathbf{S} + \mathbf{E}$. Otherwise, it samples $\mathbf{G} \leftarrow \mathbb{Z}_q^{2n \cdot \ell_{\mathrm{PRG}} \times (m-n)}$. $\mathcal{C}$ finally sends the LWE-ss challenge matrices $(\mathbf{H}, \mathbf{G})$ to $\mathcal{B}$. $\mathcal{B}$ partitions $\mathbf{H}$ into $2 \cdot \ell_{\mathrm{PRG}}$ submatrices $(\mathbf{H}^{(1)}, \mathbf{H}^{(2)}, \ldots, \mathbf{H}^{(2\ell_{\mathrm{PRG}})})$ each of dimension $n \times n$. Next, it partitions $\mathbf{G}$ into $2 \cdot \ell_{\mathrm{PRG}}$ submatrices $(\mathbf{G}^{(1)}, \mathbf{G}^{(2)}, \ldots, \mathbf{G}^{(2\ell_{\mathrm{PRG}})})$ each of dimension $n \times (m-n)$. $\mathcal{B}$ then receives challenge program P and challenge message msg from the adversary $\mathcal{A}$. Next, it chooses LHE keys, computes the ciphertext and samples matrices $\left\{ \mathbf{B}_j^{(i)} \right\}_{i \leq \ell_{\mathrm{PRG}}, j < L}$ as in the two games. Now, it needs to choose the top level matrices $\left\{ \mathbf{B}_{L,k}^{(i)} \right\}_i$. It chooses the matrices as follows.

$$\mathbf{B}^{(i)}_{L,\mathsf{rej}^{(i)}} = \left[\mathbf{H}^{(2i)} \,||\, \mathbf{G}^{(2i)} + \beta_i \cdot \mathbf{D} \right]$$

$$\mathbf{B}^{(i)}_{L,\mathsf{acc}^{(i)}} = \left[\mathbf{H}^{(2i-1)} \,||\, \mathbf{G}^{(2i-1)} + (1-\beta_i) \cdot \mathbf{D} \right]$$

$$\mathbf{B}^{(i)}_{L,k} \leftarrow \mathbb{Z}_q^{n \times m} \text{ if } k \notin \{\mathsf{acc}^{(i)}, \mathsf{rej}^{(i)}\}$$

where $\mathbf{D} = q^{3/4} \cdot \left[\mathbf{I}_n \,||\, \mathbf{0}^{n \times (m - 2 \cdot n)} \right]$. $\mathcal{B}$ then samples the matrices $\left\{ \mathbf{C}_j^{(i,0)}, \mathbf{C}_j^{(i,1)} \right\}_{j<L}$ as in the two games. Finally, $\mathcal{B}$ sends the obfuscated program which consists of the LHE evaluation key, LHE ciphertext, together with the components $\left(\left\{ \mathbf{B}_{0,1}^{(i)} \right\}_i, \left\{ (\mathbf{C}_j^{(i,0)}, \mathbf{C}_j^{(i,1)}) \right\}_{i,j} \right)$ to the adversary. $\mathcal{A}$ outputs a bit b', which $\mathcal{B}$ outputs as its guess in LWE-ss game.

Note that $\mathcal{B}$ simulates Game 2 to $\mathcal{A}$ if $b = 0$, and simulates Game 3 to $\mathcal{A}$ if $b = 1$. Therefore, the advantage of $\mathcal{B}$ in LWE-ss security game is non-negligible. ∎

Lemma 4.5. For any adversary $\mathcal{A}$, $p_3^{\mathcal{A}} = p_4^{\mathcal{A}}$.

Proof. There is only a syntactic change between Games 3 and 4. The distribution of matrices generated by the challenger in Game 3 and Game 4 are identical. ∎

Lemma 4.6. For any adversary $\mathcal{A}$, there exists a negligible function negl$(\cdot)$ such that for all $\lambda \in \mathbb{N}$, we have $|p_4^{\mathcal{A}} - p_5^{\mathcal{A}}| \leq \mathrm{negl}(\lambda)$.

Proof. The proof of this claim is similar to the proof of in [GKW17a, Claim 4.6]. ∎

Lemma 4.7. Assuming the security of PRG, for any PPT adversary A, there exists a negligible function $\text{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, we have $|p_5^{\mathcal{A}} - p_6^{\mathcal{A}}| \leq \text{negl}(\lambda)$.

Proof. This proof is similar to proof of Claim 4.2. ■

By combining the above lemmas, our construction is computationally indistinguishable from [GKW17a, Appendix D] construction that uses perfectly injective PRGs. We note that Goyal et al. prove the following theorem.

Theorem 4.3. [GKW17a] (Appendix D, Parapharased): Assuming that LHE is a secure leveled homomorphic encryption scheme, PRG is a secure perfectly injective pseudorandom generator, lattice trapdoors are secure and $(n, 5m \cdot \ell_{\text{PRG}}, n, q, \chi)$-$\mathsf{LWE}$-$\mathsf{ss}$ assumptions hold, the lockable obfuscation construction described in [GKW17a, Appendix D] is secure as per Definition 2.2.

Combining theorems 4.2 and 4.3, we obtain theorem 4.1. ■

5 Perfectly Injective PRGs from LPN

In this section, we give our construction of (perfectly) injective PRGs (with Setup) from the Learning Parity with Noise assumption.[12]

Overview. Let the input length of PRG be $n+\ell$. We parse input $x \in \{0,1\}^{n+\ell}$ as $x = y \,||\, z$, where $|y| = n$ and $|z| = \ell$. Now, string y is parsed as $\mathbf{s}$, and z will be used to sample the error vector $\mathbf{e}$. Note that for injectivity argument to go through, it is important that the mapping between input y, z and vectors $\mathbf{s}, \mathbf{e}$ is also injective. Now both y and $\mathbf{s}$ are already of length n, thus we only need to make sure that our error vector sampling procedure is injective. Before describing our sampling procedure, we would like to point out that, in the PRG security game, the PRG seed is sampled uniformly at random, thus the distribution over error vectors will be a uniform distribution as well. This suggests that for basing pseudorandomness security we can't rely on the standard LPN assumption as the noise distribution is not Bernoulli, but uniform. However, we could instead rely on the exact-LPN assumption (or xLPN) which is polynomially related to standard LPN assumption, and in which the noise distribution is uniform as the error vectors are sampled such that they have fixed hamming weight.

Next, we observe that the size of support of noise distribution in the the xLPN assumption need not be a perfect *power of two*, thus we might not be able to injectively sample error vectors from the fixed length binary string z. To resolve this issue, we simply truncate the noise distribution to contain only lexically smallest error vectors such that the size of truncated set is equal to the nearest power of two. However, with this modification we need to rely on an alternate assumption which we call the restricted-exact-LPN assumption (or rxLPN). It turns out that the sample-preserving reduction of [AIK09] also holds for rxLPN. This suggests that rxLPN and LPN assumptions are (polynomially) equivalent, therefore we could still reduce the security to the LPN assumption. Now to injectively map vectors with a fixed hamming weight to bitstrings, we employ a simple combinatorial trick to give a total ordering over vectors with efficient recursive sampling procedure. First, note that a total ordering over vectors can be trivially defined by denoting each vector with its corresponding integer representation. Now, our sampling procedure works as follows — let $x \in \{0,1\}^{\ell}$ and we want to sample vector $\mathbf{v} \in \mathbb{Z}_2^m$ such that $\mathsf{HW}(\mathbf{v}) = k$. The sampling algorithm first checks whether $\mathsf{int}(x) > {}^{m-1}C_k$ (where $\mathsf{int}(x)$ is the integer corresponding to string x). If the check succeeds, then it sets the first position in $\mathbf{v}$ to be 1, else it sets it 0, and continues. Also, if the check succeeds, then it updates $x = x - {}^{m-1}C_k$. In other words, each vector $\mathbf{v} \in \mathbb{Z}_2^m$ with $\mathsf{HW}(\mathbf{v}) = k$ is uniquely ranked from 0 to ${}^{m}C_k - 1$,

[12] Our PRG construction bears some resemblance to the IND-CCA secure encryption schemes provided by Döttling et al. [DMQN12] and Kiltz et al. [KMP14], but requires new ideas. We point that if we try to build PRGs using the techniques from [DMQN12, KMP14], then that only gives 'statistically injective' PRGs, whereas in this paper our goal is to get *perfectly injective* PRGs.

and the sample algorithm outputs vector $\mathbf{v}$ with rank $\mathsf{int}(x)$. For example, $0^{m-k}1^k$ has rank 0 and 1^k0^{m-k} has rank ${}^mC_k - 1$. The sampling procedure has been formally described later in Algorithm 1.

Finally, to sample matrix $\mathbf{B}$ as a generator matrix of some good but random code, we employ ideas similar to that used in our LWE solution. To sample $\mathbf{B}$ in this special way, we simply choose a uniformly random matrix $\mathbf{A}$, a matrix $\mathbf{C}$ *with low hamming weight rows* and set $\mathbf{B} = [\mathbf{A} \mid \mathbf{AC} + \mathbf{G}]$, where $\mathbf{G}$ is the generator matrix of an error correcting code. Here the role of $\mathbf{G}$ is similar to the role of $\mathbf{D}$ in the previous solution, that is to map any non-zero vector to a high hamming weight vector. A crucial point here is that the rows of $\mathbf{C}$ must have low hamming weight. This is because if $\mathbf{A}^T\mathbf{s}$ has low hamming weight, then so does $\mathbf{C}^T\mathbf{A}^T\mathbf{s}$, and later this will be crucial in arguing that $\mathbf{B}$ is a generator matrix of a good code. Finally, for pseudorandomness of our construction, we want that $\mathbf{B}$ should look like a random matrix to any computationally bounded adversary. To this end, we use the Knapsack LPN assumption which was also shown to be (polynomially) equivalent to LPN assumption [MM11].[13] This is discussed in detail in Section 5.

Before formally describing our construction, we define a (bijective) sampling procedure Sample that takes as input a length ℓ bit string s and outputs a (unique) vector $\mathbf{v} \in \chi^{(\mathsf{re})}_{k,\tau}$, where $|\chi^{(\mathsf{re})}_{k,\tau}| = 2^\ell$. In other words, we describe a poly-time procedure to injectively sample noise vectors as per rxLPN noise distribution. A similar lexicographic ordering was first considered by Fischer and Stern [FS96].

Algorithm 1 Procedure for Injectively Sampling Error Vectors

```
function Sample($s \in \{0,1\}^\ell$)  $\to \mathbf{v} \in \chi^{(\mathsf{re})}_{k,\tau}$
    Set $\mathsf{index} = \mathsf{int}(s)$ and $n = \lfloor k\tau \rceil$
    for all $i \in \{1 \ldots k\}$ do
        if $\mathsf{index} > {}^{k-i}C_{n-1}$ then
            Set $v_{k-i+1} = 1$, $\mathsf{index} = \mathsf{index} - {}^{k-i}C_{n-1}$, $n = n - 1$
        else if $\mathsf{index} < {}^{k-i}C_{n-1}$ then
            Set $v_{k-i+1} = 0$
        else
            Set $v_j = 1$ for all $j \leq n$, and $v_j = 0$ for all $n < j \leq k - i + 1$
            return $(v_1, \ldots, v_k)^T$
        end if
    end for
    return $(v_1, \ldots, v_k)^T$
end function
```

We will now describe our construction. Let $\beta = 1/(c_1\sqrt{n})$ and $\chi = \mathsf{Ber}_\beta$ where c_1 is some constant. Let $\{\mathbf{G}_n \in \mathbb{Z}_2^{n\times k}\}_{n\in\mathbb{N}}$ be a family of generator matrices for error correcting codes where the distance of the code[14] generated by $\mathbf{G}_n$ is at least $c_4 \cdot n$ where $c_4 > 2$. Let $m = c_2 n, k = c_3 n$ where c_2, c_3 are any constants such that $c_1 > 2 \cdot (c_2 + c_3)$. Let $|\chi^{(\mathsf{re})}_{m+k,\beta}| = 2^\ell$.

An important point to note here is that the Bernoulli parameter needs to be $O(1/\sqrt{n})$. This is necessary for proving perfect injectivity. Recall, in the LWE perfect injectivity proof, we argue that since $\mathbf{A}^T\mathbf{s}$ has low norm, $\mathbf{C}^T\mathbf{A}^T\mathbf{s}$ also has low norm. For the analogous argument to work here, the error distribution must be $O(1/\sqrt{n})$. For instance, if the error distribution has hamming weight fraction at most $1/10\sqrt{n}$ and each row of $\mathbf{C}$ has hamming weight fraction at most $1/10\sqrt{n}$, then we can argue that $\mathbf{C}^T\mathbf{A}^T\mathbf{s}$ has hamming weight fraction at most $1/100$. If the noise rate was constant, then we cannot get an upper bound on the hamming weight fraction of $\mathbf{C}^T\mathbf{A}^T\mathbf{s}$. Below we describe our construction in detail.

$\mathsf{Setup}(1^n)$: The setup algorithm chooses random matrices $\mathbf{A} \leftarrow \mathbb{Z}_2^{n\times m}$ and $\mathbf{C} \leftarrow \chi^{m\times k}$. If there exists some row $\mathbf{c}_i$ of matrix $\mathbf{C}$ such that $\mathsf{HW}(\mathbf{c}_i) > 2k\beta$, it sets $\mathbf{B} = [\mathbf{A} \mid \mathbf{G}]$. Otherwise, it sets $\mathbf{B} = [\mathbf{A} \mid \mathbf{AC} + \mathbf{G}]$.

[13] The Knapsack LPN assumption states that for a uniformly random matrix $\mathbf{A}$ and a matrix $\mathbf{E}$ such that each entry is 1 with probability p and $\mathbf{A}$ has fewer rows than columns, then $(\mathbf{A}, \mathbf{AE})$ look like uniformly random matrices.

[14] Distance of a code is the minimum hamming weight of all non-zero codewords.

Finally, it outputs $\mathbf{B}$ as the PRG parameters.

$\mathrm{PRG}(\mathbf{B}, x \in \{0,1\}^{n+\ell})$: Let $x = y \,||\, z$, where $|y| = n$ and $|z| = \ell$. The PRG evaluation algorithm samples the error vector $\mathbf{e} \in \mathbb{Z}_2^{m+k}$ as $\mathbf{e} = \mathsf{Sample}(z)$. It interprets bit string y as a vector $\mathbf{s} \in \mathbb{Z}_2^n$. Finally, it outputs $\mathbf{v} = \mathbf{B}^T\mathbf{s} + \mathbf{e}$.

Depth of PRG **Evaluation Circuit and** PRG **Stretch.** First, note that the the PRG evaluation circuit needs to first sample the error vector $\mathbf{e}$ given the input vector $\mathbf{x}$, and then it performs a single matrix-vector multiplication. Here the sampling algorithm can easily be implemented by an $\mathbf{NC}^1$[15], and a matrix-vector multiplication can be done in can be implemented in $\mathbf{TC}^0$. Thus, the overall PRG evaluation can be easily performed by a $\mathbf{NC}^1$ circuit. Next, note that the input length in the above construction is $n + \ell$ and the output length is $m + k = (c_2 + c_3)n$. We know that $\ell = \lfloor \log_2 {}^{m+k}C_{\lfloor (m+k)\beta \rceil} \rfloor$. Since $\log_2 {}^{m+k}C_{\lfloor (m+k)\beta \rceil} < \lfloor (m+k)\beta \rceil \cdot \log_2(2e/\beta)$, we have that $\ell = O(\sqrt{n} \cdot \log_2 n)$ and thus $n + \ell < 2n$. Thus, the stretch provided by the above construction is $(c_2 + c_3)/2 = O(1)$. Thus, the above construction gives a PRG that provides a constant stretch with an $\mathbf{NC}^1$ evaluation circuit. One could increase the stretch to an arbitrary polynomial amount by self-composition, but that would increase the depth of the evaluation circuit.

Next, we prove the following theorem where we first show that our PRG construction satisfies perfect injectivity property, and later argue the pseudorandomness property for the same.

Theorem 5.1. If Knapsack Learning Parity with Noise assumption $\mathsf{KLPN}_{n,m,\beta}$ (Assumption 5) and Restricted Exact Learning Parity with Noise assumption $\mathsf{rxLPN}_{n,m,\beta}$ (Assumption 7) hold, then the above construction is a perfectly injective PRG.

5.1 Perfect Injectivity

First, we will argue perfect injectivity of the above PRG. For any input length n, constants c_1, c_2, c_3, c_4 such that $c_1 > 2 \cdot (c_2 + c_3)$ and $c_4 > 3$, any random matrix $\mathbf{A} \leftarrow \mathbb{Z}_2^{n \times m}$, any error correcting code generator matrix $\mathbb{G}_n$ with distance $> c_4 \cdot n$, and any matrix $\mathbf{C} \leftarrow \chi^{m \times k}$, consider the following two cases.

Case 1: $\mathsf{HW}(\mathbf{c}_i) > 2k\beta$ *for some row* $\mathbf{c}_i$ *of* $\mathbf{C}$. The setup algorithm sets matrix $\mathbf{B} = [\mathbf{A} \mid \mathbf{G}]$. Suppose there exists inputs $x_1, x_2 \in \{0,1\}^{n+\ell}$ such that $\mathrm{PRG}(\mathbf{B}, x_1) = \mathrm{PRG}(\mathbf{B}, x_2)$ and $x_1 \neq x_2$.

Let $x_i = y_i \,||\, z_i$ and $\mathbf{e}_i = \mathsf{Sample}(z_i)$ for $i = 1, 2$. Since $x_1 \neq x_2$, therefore either $y_1 \neq y_2$ or $z_1 \neq z_2$. We will first consider the case that $y_1 \neq y_2$. Let $\boldsymbol{\delta}\mathbf{e} = \mathbf{e}_1 - \mathbf{e}_2$ and $\boldsymbol{\delta}\mathbf{s} = \mathbf{s}_1 - \mathbf{s}_2$. Since $y_1 \neq y_2$, therefore their corresponding secret vectors $\mathbf{s}_1$ and $\mathbf{s}_2$ will also be distinct, i.e. $\boldsymbol{\delta}\mathbf{s} \neq \mathbf{0}$. We know that $\mathrm{PRG}(\mathbf{B}, x_i) = [\mathbf{A} \mid \mathbf{G}]^T \mathbf{s}_i + \mathbf{e}_i$. Since $\mathrm{PRG}(\mathbf{B}, x_1) = \mathrm{PRG}(\mathbf{B}, x_2)$, we can write that $[\mathbf{A} \mid \mathbf{G}]^T \boldsymbol{\delta}\mathbf{s} = \boldsymbol{\delta}\mathbf{e}$. By construction, we know that hamming weights of error vectors is exactly $\lfloor (m+k)\beta \rceil$. Thus, $\mathsf{HW}(\boldsymbol{\delta}\mathbf{e}) \leq 2 \cdot \lfloor (m+k)\beta \rceil$. Also, we know that $\mathsf{HW}(\mathbf{B}^T\boldsymbol{\delta}\mathbf{s}) \geq \mathsf{HW}(\mathbf{G}^T\boldsymbol{\delta}\mathbf{s}) \geq c_4 \cdot n$. Since $2 \cdot \lfloor (m+k)\beta \rceil \leq 2 \cdot (c_2 + c_3)\sqrt{n}/c_1 < c_4 \cdot n$, therefore this results in a contradiction. Thus, $\boldsymbol{\delta}\mathbf{s} = \mathbf{0}$.

Now $\boldsymbol{\delta}\mathbf{s} = \mathbf{0}$ but $z_1 \neq z_2$. In this case, we can claim that $\boldsymbol{\delta}\mathbf{e} \neq \mathbf{0}$ as this follows from the construction of our sampling algorithm. Since $\mathrm{PRG}(\mathbf{B}, x_1) = \mathrm{PRG}(\mathbf{B}, x_2)$, we can write that $[\mathbf{A} \mid \mathbf{G}]^T \boldsymbol{\delta}\mathbf{s} = \boldsymbol{\delta}\mathbf{e}$. Since $\boldsymbol{\delta}\mathbf{s} = \mathbf{0}$ but $\boldsymbol{\delta}\mathbf{e} \neq \mathbf{0}$, this results in a contradiction. Hence, we can conclude that in this case, our construction satisfies perfect injectivity.

Case 2: $\mathsf{HW}(\mathbf{c}_i) \leq 2k\beta$ *for all rows* $\mathbf{c}_i$ *of* $\mathbf{C}$. The setup algorithm sets matrix $\mathbf{B} = [\mathbf{A} \mid \mathbf{AC} + \mathbf{G}]$. Suppose there exists inputs $x_1, x_2 \in \{0,1\}^{n+\ell}$ such that $\mathrm{PRG}(\mathbf{B}, x_1) = \mathrm{PRG}(\mathbf{B}, x_2)$ and $x_1 \neq x_2$.

As before, it will be that either $y_1 \neq y_2$ or $z_1 \neq z_2$, where $x_i = y_i \,||\, z_i$ for $i = 1, 2$. Again we first consider that $y_1 \neq y_2$. Let $\mathbf{e}_i = \mathsf{Sample}(z_i)$, $\boldsymbol{\delta}\mathbf{e} = \mathbf{e}_1 - \mathbf{e}_2$ and $\boldsymbol{\delta}\mathbf{s} = \mathbf{s}_1 - \mathbf{s}_2$. Since $y_1 \neq y_2$, we have that $\boldsymbol{\delta}\mathbf{s} \neq \mathbf{0}$. We know that $\mathrm{PRG}(\mathbf{B}, x_i) = [\mathbf{A} \mid \mathbf{AC} + \mathbf{G}]^T \mathbf{s}_i + \mathbf{e}_i$. Since $\mathrm{PRG}(\mathbf{B}, x_1) = \mathrm{PRG}(\mathbf{B}, x_2)$, we can

[15] We believe that one could also do sampling more efficiently by a $\mathbf{TC}^0$ circuit. However, we leave exact analysis for future work.

write that $[\mathbf{A} \mid \mathbf{AC}+\mathbf{G}]^T \boldsymbol{\delta}\mathbf{s} = \boldsymbol{\delta}\mathbf{e}$. By construction, we know that hamming weights of error vectors is exactly $\lfloor (m+k)\beta \rceil$. Thus, $\mathsf{HW}(\boldsymbol{\delta}\mathbf{e}) \leq 2 \cdot \lfloor (m+k)\beta \rceil$. Therefore, $\mathsf{HW}([\mathbf{A} \mid \mathbf{AC}+\mathbf{G}]^T \boldsymbol{\delta}\mathbf{s}) \leq 2 \cdot \lfloor (m+k)\beta \rceil$.

This implies, in particular, $\mathsf{HW}(\mathbf{A}^T \boldsymbol{\delta}\mathbf{s}) \leq 2 \cdot \lfloor (m+k)\beta \rceil < n$. Also, since each row of $\mathbf{C}$ has hamming weight at most $2k\beta$, we have that $\mathsf{HW}((\mathbf{AC})^T \boldsymbol{\delta}\mathbf{s}) \leq 4k\beta \lfloor (m+k)\beta \rceil \leq 4 \cdot c_3(c_2+c_3)n/c_1^2 < n$. As a result, $\mathsf{HW}([\mathbf{A} \mid \mathbf{AC}]^T \boldsymbol{\delta}\mathbf{s}) < 2n$. But since $\boldsymbol{\delta}\mathbf{s} \neq \mathbf{0}$, we have $\mathsf{HW}(\mathbf{G}^T \boldsymbol{\delta}\mathbf{s}) \geq c_4 \cdot n$. Thus, using triangle inequality, we have that $\mathsf{HW}(\mathbf{B}^T \boldsymbol{\delta}\mathbf{s}) > c_4 \cdot n - 2n > n$. This brings us to a contradiction since $\mathsf{HW}(\boldsymbol{\delta}\mathbf{e}) < n$. Thus, $\boldsymbol{\delta}\mathbf{s} = \mathbf{0}$.

Now we have that $\boldsymbol{\delta}\mathbf{s} = \mathbf{0}$. If $z_1 \neq z_2$ (i.e., $\boldsymbol{\delta}\mathbf{e} \neq \mathbf{0}$), then by the same argument as used in *Case 1*, we can conclude that $\mathrm{PRG}(\mathbf{B}, x_1) = \mathrm{PRG}(\mathbf{B}, x_2)$ implies $x_1 = x_2$. Hence, we can conclude that our construction satisfies perfect injectivity. This concludes our proof.

5.2 Pseudorandomness

At a high level, the pseudorandomness proof proceeds as follows. First, we will first switch $\mathbf{B}$ to a uniformly random matrix during setup phase. This will follow from Knapsack LPN (KLPN) with low noise assumption. Next, we will simply switch the PRG output $\mathbf{v}$ to a uniformly random bit vector. For this step, we will use our restricted-exact LPN (rxLPN) with low noise assumption.[16] We will now argue this formally via a sequence of hybrids.

- **Hybrid 0**: This corresponds to the real world in which the challenger honestly generates matrix $\mathbf{B}$ during setup, chooses a uniformly random bit string $x \in \{0,1\}^{n+\ell}$, and computes $\mathbf{v}_0 = \mathrm{PRG}(\mathbf{B}, x)$. It chooses a random bit b and vector $\mathbf{v}_1 \leftarrow \mathbb{Z}_2^{m+k}$. It sends $(\mathbf{B}, \mathbf{v}_b)$ to the adversary.
- **Hybrid 1**: This hybrid is identical to the previous one, except that the challenger does not check if rows of matrix $\mathbf{C}$ have low hamming weight, instead it always sets $\mathbf{B}$ as $[\mathbf{A} \mid \mathbf{AC}+\mathbf{G}]$.

  It chooses random matrices $\mathbf{A} \leftarrow \mathbb{Z}_2^{n \times m}$, $\mathbf{C} \leftarrow \chi^{m \times k}$, and sets $\mathbf{B} = [\mathbf{A} \mid \mathbf{AC}+\mathbf{G}]$. Next, it chooses secret vector $\mathbf{s} \leftarrow \mathbb{Z}_2^n$, error vector $\mathbf{e} \leftarrow \chi_{m+k,\beta}^{(\mathsf{re})}$, and sets $\mathbf{v}_0 = \mathbf{B}^T\mathbf{s} + \mathbf{e}$. It chooses a random bit b and vector $\mathbf{v}_1 \leftarrow \mathbb{Z}_2^{m+k}$. Finally, it sends $(\mathbf{B}, \mathbf{v}_b)$ to the adversary.
- **Hybrid 2**: In this hybrid, the challenger simply chooses $\mathbf{B}$ uniformly at random.

  It chooses random matrices $\mathbf{B} \leftarrow \mathbb{Z}_2^{n \times (m+k)}$, secret vector $\mathbf{s} \leftarrow \mathbb{Z}_2^n$, error vector $\mathbf{e} \leftarrow \chi_{m+k,\beta}^{(\mathsf{re})}$, and sets $\mathbf{v}_0 = \mathbf{B}^T\mathbf{s} + \mathbf{e}$. It chooses a random bit b and vector $\mathbf{v}_1 \leftarrow \mathbb{Z}_2^{m+k}$. Finally, it sends $(\mathbf{B}, \mathbf{v}_b)$ to the adversary.
- **Hybrid 3**: In this hybrid, the challenger chooses $\mathbf{v}_0$ uniformly at random as well.

  It chooses random matrices $\mathbf{B} \leftarrow \mathbb{Z}_2^{n \times (m+k)}$ and vector $\mathbf{v} \leftarrow \mathbb{Z}_2^{m+k}$. Finally, it sends $(\mathbf{B}, \mathbf{v})$ to the adversary.

Let $\mathsf{Adv}_i^{\mathcal{A}}$ denote the advantage of adversary $\mathcal{A}$ in Hybrid i. We will now show that for all $i \in \{0,1,2\}$, $\mathsf{Adv}_i^{\mathcal{A}} - \mathsf{Adv}_{i+1}^{\mathcal{A}}$ is negligible in n.

Lemma 5.1. For any adversary $\mathcal{A}$, $\mathsf{Adv}_0^{\mathcal{A}} - \mathsf{Adv}_1^{\mathcal{A}} \leq \mathrm{negl}(n)$.

Proof. The only difference between Hybrid 0 and Hybrid 1 is in the way the challenger sets $\mathbf{B}$ if some row of $\mathbf{C}$ has hamming weight greater than $2k\beta$. Since each entry of matrix $\mathbf{C}$ is sampled from a Bernoulli distribution with parameter $\beta = 1/(c_1\sqrt{n})$ (for some constant c_1), using Chernoff bounds, we can argue that $\Pr[\exists\ i \leq m \text{ such that } \mathsf{HW}(\mathbf{c}_i) > 2k\beta] \leq \mathrm{negl}(n)$. ∎

Lemma 5.2. Assuming the Knapsack Learning Parity with Noise assumption holds for $\beta = 1/(c_1\sqrt{n})$, then for any PPT adversary $\mathcal{A}$, $\mathsf{Adv}_1^{\mathcal{A}} - \mathsf{Adv}_2^{\mathcal{A}} \leq \mathrm{negl}(n)$.

[16] Recall that rxLPN is equivalent to standard LPN assumption.

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ such that $\mathsf{Adv}_1^{\mathcal{A}} - \mathsf{Adv}_2^{\mathcal{A}} = \epsilon$. We will construct a reduction algorithm $\mathcal{B}$ that breaks the knapsack LPN assumption with advantage ϵ. The reduction algorithm $\mathcal{B}$ receives matrices $\mathbf{X} \in \mathbb{Z}_2^{n\times m}$, $\mathbf{Y} \in \mathbb{Z}_2^{n\times k}$ where $\mathbf{Y}$ is either a uniformly random matrix, or $\mathbf{Y} = \mathbf{XZ}$ for some matrix $\mathbf{Z} \leftarrow \mathsf{Ber}_\beta^{m\times k}$.[17] It sets $\mathbf{A} = \mathbf{X}$, $\mathbf{B} = [\mathbf{A} \mid \mathbf{Y} + \mathbf{G}]$, and chooses $\mathbf{s} \leftarrow \mathbb{Z}_2^n$, $\mathbf{e} \leftarrow \chi^{(\mathsf{re})}_{m+k,\beta}$, and sets $\mathbf{v}_0 = \mathbf{B}^T\mathbf{s} + \mathbf{e}$. It chooses a random bit b and vector $\mathbf{v}_1 \leftarrow \mathbb{Z}_2^{m+k}$. Finally, it sends $(\mathbf{B}, \mathbf{v}_b)$ to the adversary. Finally, if the adversary guesses bit b correctly, then $\mathcal{B}$ guesses that $\mathbf{Y} = \mathbf{XZ}$, otherwise it guesses $\mathbf{Y}$ is a uniformly random matrix.

The algorithm $\mathcal{B}$ thus breaks the Knapsack LPN assumption with advantage ϵ. ■

Lemma 5.3. Assuming the Restricted-Exact LPN assumption holds for $\beta = 1/(c_1\sqrt{n})$, then for any PPT adversary $\mathcal{A}$, $\mathsf{Adv}_2^{\mathcal{A}} - \mathsf{Adv}_3^{\mathcal{A}} \leq \text{negl}(n)$.

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ such that $\mathsf{Adv}_2^{\mathcal{A}} - \mathsf{Adv}_3^{\mathcal{A}} = \epsilon$. We will construct a reduction algorithm $\mathcal{B}$ that breaks the rxLPN assumption with advantage ϵ. The reduction algorithm $\mathcal{B}$ receives matrices $\mathbf{B} \in \mathbb{Z}_2^{n\times(m+k)}$, and a vector $\mathbf{v} \in \mathbb{Z}_2^{(m+k)}$, where $\mathbf{v}$ is either a uniformly random vector, or $\mathbf{v} = \mathbf{A}^T\mathbf{s} + \mathbf{e}$ for some vectors $\mathbf{s}, \mathbf{e}$ sampled as $\mathbf{s} \leftarrow \mathbb{Z}_2^n$, $\mathbf{e} \leftarrow \chi^{(\mathsf{re})}_{m+k,\beta}$. $\mathcal{B}$ sends $(\mathbf{B}, \mathbf{v})$ to the adversary. Finally, $\mathcal{B}$ forwards the adversary's guess as its own guess.

The algorithm $\mathcal{B}$ thus breaks the Restricted-Exact LPN assumption with advantage ϵ. ■

Finally, note that any adversary has 0 advantage in the hybrid 3. From the above lemmas, it follows that under the LPN with low noise assumption, the PRG construction is secure.

References

[ACPS09] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In *CRYPTO*, pages 595–618, 2009.

[AG11] Sanjeev Arora and Rong Ge. New algorithms for learning in presence of errors. In *International Colloquium on Automata, Languages, and Programming*, pages 403–415. Springer, 2011.

[AIK09] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography with constant input locality. *Journal of Cryptology*, 22(4):429–469, 2009.

[Ajt99] Miklós Ajtai. Generating hard instances of the short basis problem. In *Automata, Languages and Programming, 26th International Colloquium, ICALP'99, Prague, Czech Republic, July 11-15, 1999, Proceedings*, pages 1–9, 1999.

[AKPW13] Joël Alwen, Stephan Krenn, Krzysztof Pietrzak, and Daniel Wichs. Learning with rounding, revisited. In *Advances in Cryptology–CRYPTO 2013*, pages 57–74. Springer, 2013.

[Ale03] Michael Alekhnovich. More on average case vs approximation complexity. In *Foundations of Computer Science, 2003. Proceedings. 44th Annual IEEE Symposium on*, 2003.

[Bar86] D A Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in nc1. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, STOC '86, 1986.

[BDFP86] Allan Borodin, Danny Dolev, Faith E. Fich, and Wolfgang J. Paul. Bounds for width two branching programs. *SIAM J. Comput.*, 15(2):549–560, 1986.

[17] Note that the standard Knapsack-LPN states that matrices $\mathbf{Y}, \mathbf{Z}$ will be square matrices. However, here we consider non-square matrices as well. We would like to point out this non-square version is implied from Knapsack-LPN by a standard hybrid argument.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. In *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '01, 2001.

[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 *CRYPTO*, pages 1–18, 2001.

[BGM$^+$16] Andrej Bogdanov, Siyao Guo, Daniel Masny, Silas Richelson, and Alon Rosen. On the hardness of learning with rounding over small modulus. In *Theory of Cryptography Conference*, pages 209–224. Springer, 2016.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *ITCS*, 2012.

[BKP19] Nir Bitansky, Dakshita Khurana, and Omer Paneth. Weak zero-knowledge beyond the black-box barrier. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019.*, pages 1091–1102, 2019.

[BLP$^+$13a] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 575–584, 2013.

[BLP$^+$13b] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 575–584, 2013.

[BLSV18] Zvika Brakerski, Alex Lombardi, Gil Segev, and Vinod Vaikuntanathan. Anonymous ibe, leakage resilience and circular security from new assumptions. In *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part I*, pages 535–564, 2018.

[BLVW18] Zvika Brakerski, Vadim Lyubashevsky, Vinod Vaikuntanathan, and Daniel Wichs. Worst-case hardness for LPN and cryptographic hashing via code smoothing. *IACR Cryptology ePrint Archive*, 2018:279, 2018.

[BPR12] Abhishek Banerjee, Chris Peikert, and Alon Rosen. Pseudorandom functions and lattices. In *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, pages 719–737, 2012.

[BSW06] Dan Boneh, Amit Sahai, and Brent Waters. Fully collusion resistant traitor tracing with short ciphertexts and private keys. In *EUROCRYPT*, pages 573–592, 2006.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. In *FOCS*, pages 97–106, 2011.

[BV16] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation: From approximate to exact. In *Theory of Cryptography - 13th International Conference, TCC 2016-A*, 2016.

[BV17] Nir Bitansky and Vinod Vaikuntanathan. A note on perfect correctness by derandomization. In *Advances in Cryptology - EUROCRYPT 2017*, pages 592–606, 2017.

[CH85] Stephen A. Cook and H. James Hoover. A depth-universal circuit. *SIAM Journal on Computing*, 14(4):833–839, 1985.

[Coc01] Clifford Cocks. An identity based encryption scheme based on quadratic residues. In *IMA Int. Conf.*, pages 360–363, 2001.

[CVW$^+$18a] Yilei Chen, Vinod Vaikuntanathan, Brent Waters, Hoeteck Wee, and Daniel Wichs. Traitor-tracing from lwe made simple and attribute-based. In *TCC*, 2018.

[CVW18b] Yilei Chen, Vinod Vaikuntanathan, and Hoeteck Wee. Ggh15 beyond permutation branching programs: Proofs, attacks, and candidates. Cryptology ePrint Archive, Report 2018/360, 2018. https://eprint.iacr.org/2018/360.

[DGHM18] Nico Döttling, Sanjam Garg, Mohammad Hajiabadi, and Daniel Masny. New constructions of identity-based and key-dependent message secure encryption schemes. In *Public-Key Cryptography - PKC 2018 - 21st IACR International Conference on Practice and Theory of Public-Key Cryptography, Rio de Janeiro, Brazil, March 25-29, 2018, Proceedings, Part I*, pages 3–31, 2018.

[DMQN12] Nico Döttling, Jörn Müller-Quade, and Anderson CA Nascimento. Ind-cca secure cryptography based on a variant of the lpn problem. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 485–503. Springer, 2012.

[DORS08] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam D. Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM J. Comput.*, 38(1):97–139, 2008.

[DRS04] Yevgeniy Dodis, Leonid Reyzin, and Adam D. Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, pages 523–540, 2004.

[FS96] Jean-Bernard Fischer and Jacques Stern. An efficient pseudo-random generator provably as secure as syndrome decoding. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 245–255. Springer, 1996.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.

[GGH$^+$13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013*, pages 40–49, 2013.

[GHKW17] Rishab Goyal, Susan Hohenberger, Venkata Koppula, and Brent Waters. A generic approach to constructing and proving verifiable random functions. In *Theory of Cryptography - 15th International Conference, TCC 2017*, 2017.

[GKW17a] Rishab Goyal, Venkata Koppula, and Brent Waters. Lockable obfuscation. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017*, pages 612–621, 2017.

[GKW17b] Rishab Goyal, Venkata Koppula, and Brent Waters. Lockable obfuscation. Cryptology ePrint Archive, Report 2017/274, 2017. https://eprint.iacr.org/2017/274.

[GKW17c] Rishab Goyal, Venkata Koppula, and Brent Waters. Separating semantic and circular security for symmetric-key bit encryption from the learning with errors assumption. In *EUROCRYPT*, 2017.

[GL89] Oded Goldreich and Leonid A Levin. A hard-core predicate for all one-way functions. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 25–32. ACM, 1989.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, pages 197–206, 2008.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *CRYPTO*, 2013.

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

[JKPT12] Abhishek Jain, Stephan Krenn, Krzysztof Pietrzak, and Aris Tentes. Commitments and efficient zero-knowledge proofs from learning parity with noise. In *Proceedings of the 18th International Conference on The Theory and Application of Cryptology and Information Security*, ASIACRYPT'12, pages 663–680, Berlin, Heidelberg, 2012. Springer-Verlag.

[KMP14] Eike Kiltz, Daniel Masny, and Krzysztof Pietrzak. Simple chosen-ciphertext security from low-noise lpn. In *International Workshop on Public Key Cryptography*, pages 1–18. Springer, 2014.

[MM11] Daniele Micciancio and Petros Mol. Pseudorandom knapsacks and the sample complexity of LWE search-to-decision reductions. In *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, pages 465–484, 2011.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, pages 700–718, 2012.

[MR07] Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on gaussian measures. *SIAM J. Comput.*, 37(1):267–302, April 2007.

[NW94] Noam Nisan and Avi Wigderson. Hardness vs randomness. *J. Comput. Syst. Sci.*, 49(2):149–167, October 1994.

[Pei09] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 333–342, 2009.

[RAD78] Ron Rivest, Leonard Adleman, and Michael L. Dertouzos. On data banks and privacy homomorphisms. In *Foundations of Secure Computation*, pages 169–180, 1978.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 84–93, 2005.

[Sha85] Adi Shamir. Identity-based cryptosystems and signature schemes. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 47–53, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

[SW05] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, pages 457–473, 2005.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *STOC*, pages 475–484, 2014.

[WZ17] Daniel Wichs and Giorgos Zirdelis. Obfuscating compute-and-compare programs under lwe. In *Foundations of Computer Science (FOCS), 2017 IEEE 58th Annual Symposium on*, 2017.

[YS16] Yu Yu and John P. Steinberger. Pseudorandom functions in almost constant depth from low-noise LPN. In *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, pages 154–183, 2016.

[YZ16] Yu Yu and Jiang Zhang. Cryptography with auxiliary input and trapdoor from constant-noise LPN. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I*, pages 214–243, 2016.

[YZW$^+$17] Yu Yu, Jiang Zhang, Jian Weng, Chun Guo, and Xiangxue Li. Collision resistant hashing from learning parity with noise. *IACR Cryptology ePrint Archive*, 2017:1260, 2017.

A Lattices with Trapdoors

Lattices with trapdoors are lattices that are statistically indistinguishable from randomly chosen lattices, but have certain 'trapdoors' that allow efficient solutions to hard lattice problems.

Definition A.1 ([Ajt99, GPV08])**.** A trapdoor lattice sampler consists of algorithms $\mathsf{TrapGen}$ and $\mathsf{SamplePre}$ with the following syntax and properties:

- $\mathsf{TrapGen}(1^n, 1^m, q) \to (\mathbf{A}, T_{\mathbf{A}})$: The lattice generation algorithm is a randomized algorithm that takes as input the matrix dimensions n, m, modulus q, and outputs a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ together with a trapdoor $T_{\mathbf{A}}$.
- $\mathsf{SamplePre}(\mathbf{A}, T_{\mathbf{A}}, \mathbf{u}, \sigma) \to \mathbf{s}$: The presampling algorithm takes as input a matrix $\mathbf{A}$, trapdoor $T_{\mathbf{A}}$, a vector $\mathbf{u} \in \mathbb{Z}_q^n$ and a parameter $\sigma \in \mathcal{R}$ (which determines the length of the output vectors). It outputs a vector $\mathbf{s} \in \mathbb{Z}_q^m$.

These algorithms must satisfy the following properties:

1. *Correct Presampling:* For all vectors $\mathbf{u}$, parameters σ, $(\mathbf{A}, T_{\mathbf{A}}) \leftarrow \mathsf{TrapGen}(1^n, 1^m, q)$, and $\mathbf{s} \leftarrow \mathsf{SamplePre}(\mathbf{A}, T_{\mathbf{A}}, \mathbf{u}, \sigma)$, $\mathbf{A} \cdot \mathbf{s} = \mathbf{u}$ and $\|\mathbf{s}\|_\infty \leq \sqrt{m} \cdot \sigma$.
2. *Well Distributedness of Matrix:* The following distributions are statistically indistinguishable:
$$\{\mathbf{A} : (\mathbf{A}, T_{\mathbf{A}}) \leftarrow \mathsf{TrapGen}(1^n, 1^m, q)\} \approx_s \{\mathbf{A} : \mathbf{A} \leftarrow \mathbb{Z}_q^{n\times m}\}.$$
3. *Well Distributedness of Preimage:* For all $(\mathbf{A}, T_{\mathbf{A}}) \leftarrow \mathsf{TrapGen}(1^n, 1^m, q)$, if $\sigma = \omega(\sqrt{n \cdot \log q \cdot \log m})$, then the following distributions are statistically indistinguishable:
$$\{\mathbf{s} : \mathbf{u} \leftarrow \mathbb{Z}_q^n, \mathbf{s} \leftarrow \mathsf{SamplePre}(\mathbf{A}, T_{\mathbf{A}}, \mathbf{u}, \sigma)\} \approx_s \mathcal{D}_{\mathbb{Z}^m,\sigma}.$$

These properties are satisfied by the gadget-based trapdoor lattice sampler of [MP12] for parameters m such that $m = \Omega(n \cdot \log q)$.

B Branching Programs

Branching programs are a model of computation used to capture space-bounded computations [BDFP86, Bar86]. In this work, we will be using a restricted notion called *permutation branching programs*.

Definition B.1 (Permutation Branching Program)**.** A permutation branching program of length L, width w and input space $\{0,1\}^n$ consists of a sequence of $2L$ permutations $\sigma_{i,b} : [w] \to [w]$ for $1 \le i \le L, b \in \{0,1\}$, an input selection function $\mathsf{inp} : [L] \to [n]$, an accepting state $\mathsf{acc} \in [w]$ and a rejection state $\mathsf{rej} \in [w]$. The starting state st_0 is set to be 1 without loss of generality. The branching program evaluation on input $x \in \{0,1\}^n$ proceeds as follows:

- For $i = 1$ to L,
  - Let $\mathsf{pos} = \mathsf{inp}(i)$ and $b = x_{\mathsf{pos}}$. Compute $\mathsf{st}_i = \sigma_{i,b}(\mathsf{st}_{i-1})$.
- If $\mathsf{st}_L = \mathsf{acc}$, output 1. If $\mathsf{st}_L = \mathsf{rej}$, output 0, else output $\perp$.

In a remarkable result, Barrington [Bar86] showed that any circuit of depth d can be simulated by a permutation branching program of width 5 and length 4^d.

Theorem B.1 ([Bar86])**.** For any boolean circuit C with input space $\{0,1\}^n$ and depth d, there exists a permutation branching program BP of width 5 and length 4^d such that for all inputs $x \in \{0,1\}^n$, $C(x) = \mathsf{BP}(x)$.

This permutation property will be useful for proving security of our main construction in 4.1. We will also require that the permutation branching program has a fixed input-selector function inp. In our construction, we will have multiple branching programs, and all of them must read the same input bit at any level $i \le L$.

Definition B.2. A permutation branching program with input space $\{0,1\}^n$ is said to have a fixed input-selector $\mathsf{inp}(\cdot)$ if for all $i \le L$, $\mathsf{inp}(i) = i \bmod n$.

Any permutation branching program of length L and input space $\{0,1\}^n$ can be easily transformed to a fixed input-selector branching program of length $n \cdot L$. In this work, we only require that all branching programs share the same input selector function $\mathsf{inp}(\cdot)$. The input selector which satisfies $\mathsf{inp}(i) = i \bmod n$ is just one possibility, and we stick with it for simplicity. We will use the following corollary, which follows from Theorem B.1.

Corollary B.1. For any boolean circuit C with input space $\{0,1\}^n$ and depth d, there exists a *fixed-input selector* permutation branching program BP of width 5 and length $n \cdot 4^d$ such that for all inputs $x \in \{0,1\}^n$, $C(x) = \mathsf{BP}(x)$.

C Homomorphic Encryption

Homomorphic encryption [RAD78, Gen09] is a powerful extension of public key encryption that allows one to evaluate functions on ciphertexts. Homomorphic encryption schemes can be classified as either leveled or fully homomorphic encryption schemes. A leveled homomorphic encryption (LHE) scheme allows bounded depth computation over the ciphertexts. The setup algorithm takes as input a 'level bound' ℓ together with the security parameter, and outputs a public-secret key pair. Given an ciphertext ct corresponding to message m, one can use the evaluation algorithm to evaluate a bounded depth circuit C on ct, and the resulting ciphertext ct', when decrypted using the secret key, outputs $C(m)$ if the depth of C is less than ℓ. Fully homomorphic encryption, on the other hand, allows for arbitrary computation on the ciphertext.

C.1 Leveled Homomorphic Encryption

A secret key leveled homomorphic encryption scheme $\mathcal{HE}$ with message space $\{0,1\}$ consists of four algorithms $\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval}$ with the following syntax:

1. $\mathsf{Setup}(1^\lambda, 1^\ell) \to (\mathsf{sk}, \mathsf{ek})$ The setup algorithm takes as input the security parameter λ, bound on circuit depth ℓ and outputs a secret key sk and evaluation key ek.

2. $\mathsf{Enc}(\mathsf{sk}, m \in \{0,1\}) \to \mathsf{ct}$ The encryption algorithm takes as input a secret key sk, message $m \in \{0,1\}$ and outputs a ciphertext ct.
3. $\mathsf{Eval}(\mathsf{ek}, C \in \mathcal{C}_\ell, \mathsf{ct}) \to \mathsf{ct}'$ The evaluation algorithm takes as input an evaluation key ek, a circuit $C \in \mathcal{C}_\ell$, a ciphertext ct and outputs a ciphertext ct'.
4. $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to x$ The decryption algorithm takes as input a secret key sk and ciphertext ct and outputs $x \in \{0,1\} \cup \{\perp\}$.

We will now define some properties of leveled homomorphic encryption schemes. Let $\mathcal{HE}$ be any homomorphic encryption scheme with message space $\{0,1\}$. First, we have the correctness property, which states that the decryption of a homomorphic evaluation on a ciphertext must be equal to the evaluation on the underlying message.

Definition C.1 (Correctness)**.** The scheme $\mathcal{HE}$ is said to be (perfectly) correct if for all security parameter λ, circuit-depth bound ℓ, $(\mathsf{sk}, \mathsf{ek}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell)$, circuit $C \in \mathcal{C}_\ell$ and message $m \in \{0,1\}$,

$$\mathsf{Dec}(\mathsf{sk}, \mathsf{Eval}(\mathsf{ek}, C, \mathsf{Enc}(\mathsf{sk}, m))) = C(m).$$

Next, we have the compactness property which requires that the size of the output of an evaluation on a ciphertext must not depend upon the evaluation circuit. In particular, we require that there exists one decryption circuit such that this circuit can decrypt any bounded-depth evaluations on ciphertexts.

Definition C.2 (Compactness)**.** A homomorphic encryption scheme $\mathcal{HE}$ is said to be compact if for all λ, ℓ there is a decryption circuit $C^{\mathsf{Dec}}_{\lambda,\ell}$ such that for all $(\mathsf{sk}, \mathsf{ek}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell)$, $m \in \{0,1\}$, $C \in \mathcal{C}_\ell$, $C^{\mathsf{Dec}}_{\lambda,\ell}(\mathsf{sk}$, $\mathsf{Eval}(\mathsf{ek}$, C, $\mathsf{Enc}(\mathsf{sk}, m))) = C(m)$.

Finally, we require that the depth of the decryption circuit is bounded by a logarithmic function in the security parameter λ.

Definition C.3. A compact homomorphic encryption scheme $\mathcal{HE}$ is said to have log-depth decryption circuit if for all λ, ℓ, $\mathsf{depth}(C^{\mathsf{Dec}}_{\lambda,\ell}) = O(\log \lambda)$.

For security, we require that the underlying scheme is IND-CPA secure.

Definition C.4 (Security)**.** A homomorphic encryption scheme $\mathcal{HE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec})$ is IND-CPA secure if for every stateful PPT adversary $\mathcal{A}$, there exists a negligible functions negl$(\cdot)$, such that the following function of λ is bounded by negl$(\cdot)$

$$\left| \Pr\left[\mathcal{A}(\mathsf{ct}) = b \ : \ \begin{array}{c} (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda); b \leftarrow \{0,1\} \\ (m_0, m_1) \leftarrow \mathcal{A}(\mathsf{pk}); \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b) \end{array} \right] - \frac{1}{2} \right|$$

Starting with the work of Gentry [Gen09], there has been a long line of interesting works seeking to improve the efficiency/security of homomorphic encryption schemes. Today, we have LHE schemes [BV11, BGV12, GSW13] with log-depth decryption circuits that can be proven secure under the Learning with Errors assumption.