On Perfect Correctness in (Lockable) Obfuscation

Rishab Goyal[1], Venkata Koppula[2], Satyanarayana Vusirikala[3], and Brent Waters[4]

[1] MIT goyal@utexas.edu
[2] Weizmann Institute of science venkata.koppula@weizmann.ac.il
[3] University of Texas at Austin satya@cs.utexas.edu
[4] University of Texas at Austin and NTT Research bwaters@cs.utexas.edu

Abstract. In a lockable obfuscation scheme [28, 39] a party takes as input a program P, a lock value α, a message msg and produces an obfuscated program $\tilde{P}$. 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) [28]. 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, a 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 could first create a program P to perform this function and then release an obfuscated version $\widetilde{P}$ that could hide features of the 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.[5] (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. This proof will certify that the program meets some specification or has some properties; e.g. "there are at most three inputs that result in the output '010'.") 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.

Another important context where perfect correctness matters is when a primitive serves as a component or building block in a larger cryptosystem. We present a few examples where a difference in perfect versus imperfect correctness in a primitive can manifest into fundamentally impacting security when complied into a larger system.

1. Dwork, Naor and Reingold [25] showed that the classical transformations of IND-CPA to IND-CCA transformations via NIZKs [33, 22] may not work when the IND-CPA scheme is not perfectly correct. They addressed this by

[5] 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.

amplifying standard imperfect correctness to what they called almost-all-keys correctness.
2. Bitansky, Khurana, and Paneth [10] constructed zero knowledge arguments with low round complexity. For their work, they required lockable obfuscation with one-sided perfect correctness.[6]
3. Recently, [4, 11] constructed constant-round post-quantum secure constant-round ZK arguments. These protocols use lockable obfuscation as a means to commit a message with pefect-binding property. Without both-sided perfect correctness, the commitment scheme and thereby the ZK argument scheme fails to be secure.

In this paper we study perfect correctness 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 [28, 39] 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 [8, 26, 37], it has been shown to be sufficient for many applications such as obfuscating conjunction and affine testers, upgrading public key encryption, identity-based encryption [38, 14, 20] and attribute-based encryption [36] to their anonymous versions and giving random oracle uninstantiatability and circular security separation results, and most recently, building efficient traitor tracing systems [15, 18].

The works of Goyal, Koppula, and Waters [28] and Wichs and Zirdelis [39] introduced and gave constructions of lockable obfuscation provably secure under the Learning with Errors [35] assumption. A limitation of both constructions (inherited from the bit-encryption cycle testers of [30]) 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 programs and identify two points in the construction that are susceptible to correctness errors. The first is in the use of an LWE-based 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.

[6] In this particular example perfect correctness [28, 39] was already present for the side they needed.

(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.

While the focus of this work has been on constructing lockable obfuscation schemes with perfect correctness building upon the schemes of [28, 39], we believe our techniques can also be applied to the recent obfuscation scheme by Chen, Vaikuntanathan, and Wee [19].

1.1 Technical Overview

We first present a short overview of the statistically correct lockable obfuscation scheme by Goyal, Koppula and Waters [29, 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.[7] 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

[7] Strictly speaking, [29, 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.

theorem [9]) that $\mathsf{BP}^{(i)}$ is of some polynomial length L and width 5.[8] 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 $\mathbf{B}^{(i)}_{j,k}$ 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 $\left\{\mathbf{B}^{(i)}_{L,1}, \ldots, \mathbf{B}^{(i)}_{L,5}\right\}$ for each $i \in [\ell_{\mathrm{PRG}}]$ uniformly at random subject to the following "sum-constraint":

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

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" $\{\mathbf{C}^{(i,0)}_j, \mathbf{C}^{(i,1)}_j\}_{i,j}$ such that each matrix $\mathbf{C}^{(i,b)}_j$ is short and can be used to evaluate its corresponding state transition permutation $\sigma^{(i)}_{j,b}$. The obfuscation of C is set to be the ℓ_{PRG} base-level matrices $\{\mathbf{B}^{(i)}_{0,1}\}_i$ and $\ell_{\mathrm{PRG}} \cdot L$ transition matrices $\{\mathbf{C}^{(i,0)}_j, \mathbf{C}^{(i,1)}_j\}_{i,j}$.

Evaluating the obfuscated program on input $x \in \{0,1\}^{\ell_{\mathrm{in}}}$ is analogous to evaluating the ℓ_{PRG} branching programs on x. For each $i \in [\ell_{\mathrm{PRG}}]$, the evaluation algorithm first computes $\mathbf{M}_i = \mathbf{B}^{(i)}_{0,1} \cdot \prod_{j=1}^{L} \mathbf{C}^{(i,x_{\mathsf{inp}(j)})}_j$ 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 $\mathbf{\Gamma} \cdot \sum_i \mathbf{B}^{(i)}_{L,\mathsf{st}^{(i)}}$ where $\mathbf{\Gamma}$ is some low-norm matrix and $\mathsf{st}^{(i)}$ denotes the final state of $\mathsf{BP}^{(i)}$.[9] It is easy to verify that if $C(x) = \alpha$, then $\widehat{C}(x) = \beta$, and therefore

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

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

[8] 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.

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

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 $x, x' \in \{0,1\}^{\ell_{\mathrm{in}}}$ 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 the top-level matrix sum for string β. That is,

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

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, Peikert and Rosen. [7], 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. [7]. 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 [27]: The perfectly injective PRG family outlined above builds upon some core ideas from the perfectly binding commitments schemes in [27]. Below, we will describe the constructions from [27], 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 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, whereas 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 [40–42, 23, 16, 17], and we feel that our perfectly injective PRGs fit this theme. Our LPN solution uses a low-noise variant ($\beta \cong \frac{1}{\sqrt{n}}$) of the LPN assumption that has been used in previous public key encryption schemes [1]. 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 taken 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. First, a recent concurrent and independent work by Asharov et al. [6] also addresses the question of perfect correctness for obfuscation. They show how to generically achieve perfect correctness for any indistinguishability obfuscation scheme, assuming hardness of LWE. Below, we discuss other related prior works.

Perfect Correctness via Derandomization. Bitansky and Vaikuntanathan [13] showed how to transform any obfuscation scheme (and a large class of cryptosystems) to remove correctness errors using Nisan-Wigderson (NW) PRGs [34]. 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 [28, 39].

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 earlier works by Bitansky and Vaikuntanathan [12], and Ananth, Jain and Sahai [3], 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. Similar correctness amplification issues were also addressed by Ananth et al.[2].

2 Preliminaries

In this section, we review the notions of injective pseudorandom generators with setup and Lockable Obfuscation [28, 39]. Due to space constraints, we review fundamentals of lattices and homomorphic encryption in the full version of the paper.

2.1 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 $\lambda \in \mathbb{N}$,

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

2.2 Lockable Obfuscation

In this section, we recall the notion of lockable obfuscation defined by Goyal et al. [28]. 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 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:*

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 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. *Let n, m, d be polynomials. A lockable obfuscation scheme* (Obf, 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. 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 $\mathcal{R} = \{0,1\}^{m \cdot \ell_{\text{out}}}$.

$\mathrm{PRG}(\mathbf{A}, \mathbf{s})$: 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.

We now prove the following theorem where we show that our PRG construction satisfies perfect injectivity property. Due to space constraints, we argue the pseudorandomness property of the construction in the full version of the paper.

Theorem 1. *If the LWR assumption with parameters n, m, p and q holds, then the above construction is a perfectly injective PRG.*

Due to space constraints, we prove the Theorem in the full version of the paper.

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. [28]. A part of the description has been taken verbatim from [28]. 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 [28].

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)

- $\ell_{\mathrm{PRG}} \cdot (L+1) \cdot (m^2 \cdot \sigma)^{L+1} < q^{1/8}$ (where $L = \ell_{\mathrm{out}} \cdot \ell_{\mathsf{ct}} \cdot 4^{d_{\mathsf{Dec}}+d_{\mathrm{PRG}}}$) (for correctness of scheme)

It is important that $L = \lambda^c$ for some constant c and $\ell_{\mathrm{PRG}} \cdot (L+1) \cdot (m^2 \cdot \sigma)^{L+1} < q^{1/8}$. 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 $\ell_{\mathrm{PRG}} = n^{3\epsilon+3}$.

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 $P \in \mathcal{C}_{\ell_{\mathrm{in}},\ell_{\mathrm{out}},d}$, message $\mathsf{msg} \in \{0,1\}$ and $\alpha \in \{0,1\}^{\ell_{\mathrm{out}}}$. The obfuscator proceeds as follows:
  1. It chooses the LHE key pair as $(\mathsf{lhe.sk}, \mathsf{lhe.ek}) \leftarrow \mathsf{LHE.Setup}(1^\lambda, 1^{d \log d})$.[10]
  2. Next, it encrypts the program P. It sets $\mathbf{ct} \leftarrow \mathsf{LHE.Enc}(\mathsf{lhe.sk}, P)$.[11]
  3. It runs $\mathsf{pp} \leftarrow \mathrm{PRG.Setup}(1^\lambda)$, and assigns $\beta = \mathrm{PRG.Eval}(\mathsf{pp}, \alpha)$.
  4. Next, consider the following circuit Q which takes as input $\ell_{\mathrm{out}} \cdot \ell_{\mathsf{ct}}$ bits of input and outputs ℓ_{PRG} bits. Q takes as input ℓ_{out} LHE ciphertexts $\{\mathsf{ct}_i\}_{i \le \ell_{\mathrm{out}}}$, 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 $\mathrm{PRG.Eval}(\mathsf{pp}, x)$. Concretely, $Q(\mathsf{ct}_1, \ldots, \mathsf{ct}_{\ell_{\mathrm{out}}}) = \mathrm{PRG.Eval}\big(\mathsf{pp}, \mathsf{LHE.Dec}(\mathsf{lhe.sk}, \mathsf{ct}_1) \,||\, \cdots \,||\, \mathsf{LHE.Dec}(\mathsf{lhe.sk}, \mathsf{ct}_{\ell_{\mathrm{out}}})\big)$.

     For $i \le \ell_{\mathrm{PRG}}$, 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 $d_{\mathsf{tot}} = d_{\mathsf{Dec}} + d_{\mathrm{PRG}}$. In the full version of the paper, we show that each branching program $\mathsf{BP}^{(i)}$ has length $L = \ell_{\mathrm{out}} \cdot \ell_{\mathsf{ct}} \cdot 4^{d_{\mathsf{tot}}}$ and width 5.
  5. The obfuscator creates matrix components which enable the evaluator to compute msg if it has an input strings (ciphertexts) $\mathsf{ct}_1, \ldots, \mathsf{ct}_{\ell_{\mathrm{out}}}$ such that $Q(\mathsf{ct}_1, \ldots, \mathsf{ct}_{\ell_{\mathrm{out}}}) = \beta$. Concretely, it runs the (randomized) routine $\mathsf{Comp\text{-}Gen}$ (defined in Figures 1, 2). 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 $\left(\left\{\mathbf{B}_{0,1}^{(i)}\right\}_i, \left\{\mathbf{C}_j^{(i,0)}, \mathbf{C}_j^{(i,1)}\right\}_{i,j}\right) \leftarrow \mathsf{Comp\text{-}Gen}(\{\mathsf{BP}^{(i)}\}_i, \beta, \mathsf{msg})$.

[10] 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.

[11] 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.

Comp-Gen

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

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

(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 $i \le \ell_{\mathrm{PRG}}$.

(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 $i \le \ell_{\mathrm{PRG}}$, $j \in [0, L-1]$, 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 $k \le 5$. 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 $i \le \ell_{\mathrm{PRG}}, k \le 5$, 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 $i \le \ell_{\mathrm{PRG}}$. It then chooses matrices $\mathbf{F}_{L,k}^{(i)}$ as follows

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

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

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

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

The algorithm continues in Figure 2.

Fig. 1: Routine Comp-Gen

6. The final obfuscated program consists of the LHE evaluation key $\mathsf{ek} = \mathsf{lhe.ek}$, LHE ciphertexts $\mathbf{ct}$, together with the components $\left(\left\{\mathbf{B}_{0,1}^{(i)}\right\}_i, \left\{(\mathbf{C}_j^{(i,0)}, \mathbf{C}_j^{(i,1)})\right\}_{i,j}\right)$.

Comp-Gen

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

i. Choose matrices $\mathbf{S}^{(0)}_{\mathsf{level}}, \mathbf{S}^{(1)}_{\mathsf{level}} \leftarrow \chi^{n \times n}$ 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 $\mathbf{I}_n$.

ii. 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 $\sigma^{(i)}_{\mathsf{level},b}(\cdot)$. More formally, for $i \leq \ell_{\mathrm{PRG}}$, set

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

iii. 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 $i \leq \ell_{\mathrm{PRG}}$.

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

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

Fig. 2: Routine Comp-Gen Continued

– $\mathsf{Eval}(\tilde{P}, x)$: 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 $x \in \{0,1\}^{\ell_{\mathrm{in}}}$. 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 [21], 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 $\ell_{\mathsf{ct}} \cdot \ell_{\mathrm{out}}$ 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 Comp-Eval (defined in Figure 3). 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 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 Comp-Gen and Comp-Eval routines. For any $z \in \{0,1\}^{\ell_{\mathrm{in}}(\lambda)}$, let $\mathsf{BP}(z) = \mathsf{BP}^{(1)}(z) \,||\, \ldots \,||\, \mathsf{BP}^{(\ell_{\mathrm{PRG}})}(z)$. Intuitively, this lemma states that for all fixed input branching programs $\{\mathsf{BP}^{(i)}\}_i$,

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 $\perp$.

Fig. 3: Routine Comp-Eval

strings β, input z, and messages msg, if $\mathsf{BP}(z) = \beta$, then the component evaluator outputs msg.

Lemma 1. *For any set of branching programs* $\{\mathsf{BP}^{(i)}\}_{i\leq \ell_{\mathrm{PRG}}}$, *string* $\beta \in \{0,1\}^{\ell_{\mathrm{PRG}}}$, *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 $\mathbf{B}_j^{(i)}$ for each $i \leq \ell_{\mathrm{PRG}}$, $j \leq L$, $\mathbf{S}_j^{(0)}, \mathbf{S}_j^{(1)}$ for each $j \leq L$ and $\mathbf{E}_j^{(i,0)}, \mathbf{E}_j^{(i,1)}$ for each $i \leq \ell_{\mathrm{PRG}}, j \leq L$. Note that the $\mathbf{S}_j^{(b)}$ and $\mathbf{E}_j^{(i,b)}$ 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.

- $\mathsf{st}_j^{(i)}$: the state of $\mathsf{BP}^{(i)}$ after j steps when evaluated on z
- $\mathbf{S}_j = \mathbf{S}_j^{(z_{\mathsf{inp}(j)})}$, $\mathbf{E}_j^{(i)} = \mathbf{E}_j^{(i,z_{\mathsf{inp}(j)})}$, $\mathbf{C}_j^{(i)} = \mathbf{C}_j^{(i,z_{\mathsf{inp}(j)})}$ for all $j \leq L$
- $\mathbf{\Gamma}_{j^*} = \prod_{j=1}^{j^*} \mathbf{S}_j$ for all $j^* \leq L$
- $\mathbf{\Delta}_{j^*}^{(i)} = \mathbf{B}_{0,1}^{(i)} \cdot \left(\prod_{j=1}^{j^*} \mathbf{C}_j^{(i)}\right)$, $\widetilde{\mathbf{\Delta}}_{j^*}^{(i)} = \mathbf{\Gamma}_{j^*} \cdot \mathbf{B}_{j^*, \mathsf{st}_{j^*}^{(i)}}^{(i)}$, $\mathbf{Err}_{j^*}^{(i)} = \mathbf{\Delta}_{j^*}^{(i)} - \widetilde{\mathbf{\Delta}}_{j^*}^{(i)}$ 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, $\mathbf{B}_x = \sum_{i:x_i=0} \mathbf{B}_{L,\mathsf{rej}^{(i)}}^{(i)} + \sum_{i:x_i=1} \mathbf{B}_{L,\mathsf{acc}^{(i)}}^{(i)}$ & $\mathbf{F}_x = \sum_{i:x_i=0} \mathbf{F}_{L,\mathsf{rej}^{(i)}}^{(i)} + \sum_{i:x_i=1} \mathbf{F}_{L,\mathsf{acc}^{(i)}}^{(i)}$ & $\mathbf{E}_x = \sum_{i:x_i=0} \mathbf{E}_{L,\mathsf{rej}^{(i)}}^{(i)} + \sum_{i:x_i=1} \mathbf{E}_{L,\mathsf{acc}^{(i)}}^{(i)}$.

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 1 *([28, Claim 4.1])* $\forall\ i \in \{1, \ldots, \ell_{\mathrm{PRG}}\}, j^* \in \{1, \ldots, L\}$, $\left\|\mathbf{Err}_{j^*}^{(i)}\right\|_\infty \leq j^* \cdot \left(m^2 \cdot \sigma\right)^{j^*}$.

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 $\mathsf{Comp\text{-}Eval}$ 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 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:

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

Note that

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

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.

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

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,

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

This implies,

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

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

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

The last inequality follows from the constraints described in the construction. Thus, matrix $\mathbf{M}$ (computed during evaluation) always satisfies the condition that $\left\|\mathbf{M}\right\|_\infty < q^{1/8}$ 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 $\left\|\mathbf{\Gamma}_L\right\|_\infty < (\sigma \cdot m^2)^L$. 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 $\left\|\mathbf{\Gamma}_L\right\|_\infty \geq 1$. 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 $\|\mathbf{E}_\beta\|_\infty < \ell_{\text{PRG}} \cdot \sigma \cdot m^2$. 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 $\widetilde{\mathbf{M}}^{(2)} = q^{1/4} \cdot \mathbf{\Gamma}_L \cdot \mathbf{S} + \mathbf{\Gamma}_L \cdot \mathbf{E}_\beta$.

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

This implies,

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

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,

$$\begin{aligned}
\|\mathbf{M}\|_\infty = \left\|\widetilde{\mathbf{M}} + \mathsf{Err}\right\|_\infty \leq \left\|\widetilde{\mathbf{M}}\right\|_\infty + \|\mathsf{Err}\|_\infty &= q^{3/8} + \mathsf{Bd} < q^{3/8} + q^{1/8} < q^{1/2}. \\
\|\mathbf{M}\|_\infty = \left\|\widetilde{\mathbf{M}} + \mathsf{Err}\right\|_\infty \geq \left\|\widetilde{\mathbf{M}}\right\|_\infty - \|\mathsf{Err}\|_\infty &\geq \left\|\widetilde{\mathbf{M}}^{(1)}\right\|_\infty - \|\mathsf{Err}\|_\infty \\
&\geq q^{1/4} - \mathsf{Bd} > q^{1/4} - q^{1/8} > q^{1/8}.
\end{aligned}$$

Hence 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 $\|\mathbf{E}_x\|_\infty \leq \ell_{\text{PRG}} \cdot \sigma \cdot m^2$ 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 $\mathbf{\Gamma}_L \cdot [\mathbf{A}_x \cdot \mathbf{S} + \mathbf{E}_x]$.

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

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 $\left\|\widetilde{\mathbf{M}}^{(2)}\right\|_\infty$.

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

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 2 *For any security parameter $\lambda \in \mathbb{N}$, any input $x \in \{0,1\}^{\ell_{\text{in}}}$, any program $P \in \mathcal{C}_{\ell_{\text{in}},\ell_{\text{out}},d}$ 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, $\mathrm{PRG.Eval}(\mathsf{pp}, \mathsf{LHE.Dec}(\mathsf{lhe.sk}, \widetilde{\mathbf{ct}})) = \beta$, where $\mathsf{pp} \leftarrow \mathrm{PRG.Setup}(1^\lambda)$.[12] Then, using Lemma 1, we can argue that $\mathsf{Comp\text{-}Eval}$ outputs msg, and thus Eval outputs msg.

Claim 3 *For all security parameters λ, inputs $x \in \{0,1\}^{\ell_{\text{in}}}$, programs $P \in \mathcal{C}_{\ell_{\text{in}},\ell_{\text{out}},d}$, $\alpha \in \{0,1\}^{\ell_{\text{out}}}$ such that $P(x) \neq \alpha$ and $\mathsf{msg} \in \{0,1\}$,*

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

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 $\mathsf{pp} \leftarrow \mathrm{PRG.Setup}(1^\lambda)$, we have $\mathrm{PRG.Eval}(\mathsf{pp}, \mathsf{LHE.Dec}(\mathsf{lhe.sk}, \widetilde{\mathbf{ct}})) \neq \beta$. Then, using Lemma 1, we can argue that $\mathsf{Comp\text{-}Eval}$ outputs $\bot$, and thus Eval outputs $\bot$.

4.3 Security

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

Theorem 2. *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.*

Proof. We prove the above theorem by proving that our construction is computationally indistinguishable from the construction provided in [28, Appendix D] that uses perfectly injective PRGs. Note that Goyal et al. [28] 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.

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

Theorem 3. *Assuming that* PRG *is a secure perfectly injective pseudorandom generator and* $(n, 2n\cdot\ell_{\mathrm{PRG}}, m-n, q, \chi)$-LWE-ss *assumption holds, the lockable obfuscation construction described in Section 4.1 is computationally indistinguishable*[13] *from [28, 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 [28]. We note that some portions of the proof are similar to those used in [28].

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 $\alpha \leftarrow \{0,1\}^{\ell_{\mathrm{out}}}$, runs $\mathsf{pp} \leftarrow \mathrm{PRG.Setup}(1^\lambda)$ and sets $\beta = \mathrm{PRG.Eval}(\mathsf{pp}, \alpha)$.
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 $\mathrm{PRG.Eval}(\mathsf{pp}, x)$. For $i \leq \ell_{\mathrm{PRG}}(\lambda)$, let $\mathsf{BP}^{(i)}$ denote the branching program that outputs the i^{th} bit of $\mathrm{PRG.Eval}(\mathsf{pp}, x)$.
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 $i \leq \ell_{\mathrm{PRG}}, k \leq 5$, uniformly at random, subject to the following constraints:

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

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

   (b) 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 $i \leq \ell_{\mathrm{PRG}}$. Next, it chooses

[13] Consider a game in which the adversary sends a program P and message msg to the challenger, which either obfuscates (P, msg) using [28] construction or our construction and sends back the obfuscated program. No PPT adversary can distinguish the two scenarios with non-negligible advantage.

matrices $\mathbf{F}^{(i)}_{L,k}$ as follows

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

(c) The top level matrices $\mathbf{B}^{(i)}_{L,k}$ for each $i \le \ell_{\mathrm{PRG}}, k \le 5$ 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 $i \le \ell_{\mathrm{PRG}}$. 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 $i \le \ell_{\mathrm{PRG}}$.
   (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 $i \le \ell_{\mathrm{PRG}}, k \le 5$, 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 $i \le \ell_{\mathrm{PRG}}$. Next, it chooses matrices $\mathbf{F}^{(i)}_{L,k}$ as follows.

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

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 $i \leq \ell_{\mathrm{PRG}}$ and $k \leq 5$, it chooses the matrices $\mathbf{B}^{(i)}_{L,k}$ uniformly at random from $\mathbb{Z}_q^{n \times m}$ 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 $\alpha \leftarrow \{0,1\}^{\ell_{\mathrm{out}}}$, runs $\mathsf{pp} \leftarrow \mathsf{PRG.Setup}(1^\lambda)$ and sets $\beta = \mathsf{PRG.Eval}(\mathsf{pp}, \alpha)$.

Due to space constraints, we prove that Game 0 is indistinguishable from Game 6 in the full version of the paper.

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.[14]

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

[14] Our PRG construction bears some resemblance to the IND-CCA secure encryption schemes provided by Döttling et al. [24] and Kiltz et al. [31], but requires new ideas. We point that if we try to build PRGs using the techniques from [24, 31], then that only gives 'statistically injective' PRGs, whereas in this paper our goal is to get *perfectly injective* PRGs.

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 [5] 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 ${}^mC_k - 1$, 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 the full version of the paper.

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 [32].[15] Due to space constraints, we defer the formal description of the construction to the full version of the paper.

[15] 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.

References

1. Alekhnovich, M.: More on average case vs approximation complexity. In: FOCS 2003 (2003)
2. Ananth, P., Jain, A., Naor, M., Sahai, A., Yogev, E.: Universal constructions and robust combiners for indistinguishability obfuscation and witness encryption. In: CRYPTO 2016 (2016)
3. Ananth, P., Jain, A., Sahai, A.: Robust transforming combiners from indistinguishability obfuscation to functional encryption. In: EUROCRYPT 2017 (2017)
4. Ananth, P., Placa, R.L.L.: Secure quantum extraction protocols. IACR Cryptol. ePrint Arch. (2019), `https://eprint.iacr.org/2019/1323`
5. Applebaum, B., Ishai, Y., Kushilevitz, E.: Cryptography with constant input locality. Journal of Cryptology 22(4) (2009)
6. Asharov, G., Ephraim, N., Komargodski, I., Pass, R.: On perfect correctness without derandomization. IACR Cryptol. ePrint Arch. 2019, 1025 (2019)
7. Banerjee, A., Peikert, C., Rosen, A.: Pseudorandom functions and lattices. In: EUROCRYPT 2012 (2012)
8. Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S.P., Yang, K.: On the (im)possibility of obfuscating programs. In: CRYPTO (2001)
9. Barrington, D.A.: Bounded-width polynomial-size branching programs recognize exactly those languages in nc1. In: STOC '86 (1986)
10. Bitansky, N., Khurana, D., Paneth, O.: Weak zero-knowledge beyond the black-box barrier. In: STOC 2019 (2019)
11. Bitansky, N., Shmueli, O.: Post-quantum zero knowledge in constant rounds. In: STOC (2020)
12. Bitansky, N., Vaikuntanathan, V.: Indistinguishability obfuscation: From approximate to exact. In: TCC 2016-A (2016)
13. Bitansky, N., Vaikuntanathan, V.: A note on perfect correctness by derandomization. In: EUROCRYPT 2017 (2017)
14. Boneh, D., Franklin, M.K.: Identity-based encryption from the Weil Pairing. In: CRYPTO '01 (2001)
15. Boneh, D., Sahai, A., Waters, B.: Fully collusion resistant traitor tracing with short ciphertexts and private keys. In: EUROCRYPT (2006)
16. Brakerski, Z., Lombardi, A., Segev, G., Vaikuntanathan, V.: Anonymous ibe, leakage resilience and circular security from new assumptions. In: EUROCRYPT 2018. pp. 535–564 (2018)
17. Brakerski, Z., Lyubashevsky, V., Vaikuntanathan, V., Wichs, D.: Worst-case hardness for LPN and cryptographic hashing via code smoothing. IACR Cryptology ePrint Archive 2018, 279 (2018)
18. Chen, Y., Vaikuntanathan, V., Waters, B., Wee, H., Wichs, D.: Traitor-tracing from lwe made simple and attribute-based. In: TCC (2018)
19. Chen, Y., Vaikuntanathan, V., Wee, H.: GGH15 beyond permutation branching programs: Proofs, attacks, and candidates. In: CRYPTO 2018 (2018)
20. Cocks, C.: An identity based encryption scheme based on quadratic residues. In: IMA Int. Conf. (2001)
21. Cook, S.A., Hoover, H.J.: A depth-universal circuit. SIAM Journal on Computing (4) (1985)
22. Dolev, D., Dwork, C., Naor, M.: Nonmalleable cryptography. SIAM J. Computing (2000)

23. Döttling, N., Garg, S., Hajiabadi, M., Masny, D.: New constructions of identity-based and key-dependent message secure encryption schemes. In: PKC 2018 (2018)
24. Döttling, N., Müller-Quade, J., Nascimento, A.C.: Ind-cca secure cryptography based on a variant of the lpn problem. In: Asiacrypt. Springer (2012)
25. Dwork, C., Naor, M., Reingold, O.: Immunizing encryption schemes from decryption errors. In: EUROCRYPT. Lecture Notes in Computer Science (2004)
26. Garg, S., Gentry, C., Halevi, S., Raykova, M., Sahai, A., Waters, B.: Candidate indstinguishability obfuscation and functional encryption for all circuits. In: FOCS (2013)
27. Goyal, R., Hohenberger, S., Koppula, V., Waters, B.: A generic approach to constructing and proving verifiable random functions. In: TCC 2017 (2017)
28. Goyal, R., Koppula, V., Waters, B.: Lockable obfuscation. In: FOCS 2017 (2017)
29. Goyal, R., Koppula, V., Waters, B.: Lockable obfuscation. Cryptology ePrint Archive, Report 2017/274 (2017), https://eprint.iacr.org/2017/274
30. Goyal, R., Koppula, V., Waters, B.: Separating semantic and circular security for symmetric-key bit encryption from the learning with errors assumption. In: EUROCRYPT (2017)
31. Kiltz, E., Masny, D., Pietrzak, K.: Simple chosen-ciphertext security from low-noise lpn. In: PKC. Springer (2014)
32. Micciancio, D., Mol, P.: Pseudorandom knapsacks and the sample complexity of LWE search-to-decision reductions. In: CRYPTO 2011 (2011)
33. Naor, M., Yung, M.: Public-key cryptosystems provably secure against chosen ciphertext attacks. In: STOC '90 (1990)
34. Nisan, N., Wigderson, A.: Hardness vs randomness. J. Comput. Syst. Sci. 49(2), 149–167 (Oct 1994), http://dx.doi.org/10.1016/S0022-0000(05)80043-1
35. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: STOC 2005 (2005)
36. Sahai, A., Waters, B.: Fuzzy identity-based encryption. In: EUROCRYPT (2005)
37. Sahai, A., Waters, B.: How to use indistinguishability obfuscation: deniable encryption, and more. In: STOC 2014 (2014)
38. Shamir, A.: Identity-based cryptosystems and signature schemes. In: CRYPTO 84
39. Wichs, D., Zirdelis, G.: Obfuscating compute-and-compare programs under LWE. In: FOCS 2017 (2017)
40. Yu, Y., Steinberger, J.P.: Pseudorandom functions in almost constant depth from low-noise LPN. In: EUROCRYPT 2016 (2016)
41. Yu, Y., Zhang, J.: Cryptography with auxiliary input and trapdoor from constant-noise LPN. In: CRYPTO 2016 (2016)
42. Yu, Y., Zhang, J., Weng, J., Guo, C., Li, X.: Collision resistant hashing from learning parity with noise. IACR Cryptology ePrint Archive 2017, 1260 (2017)

Acknowledgements. We thank anonymous reviewers for useful feedback. The work is done in part while the first and second authors were at UT Austin. The first author is supported by IBM PhD Fellowship, and the Simons Institute for the Theory of Computing (supported by Simons-Berkeley research fellowship). The second author is supported by the Simons Institute for the Theory of Computing (supported by the Simons-Berkeley research fellowship), Binational Science Foundation (Grant No. 2016726), and by the European Union Horizon 2020 Research and Innovation Program via ERC Project REACT (Grant 756482) and via Project PROMETHEUS (Grant 780701). The third author is supported by Provost Fellowship. The fourth author is supported by NSF CNS-1908611, CNS-1414082, DARPA SafeWare and Packard Foundation Fellowship.