IACR Communications in Cryptology
ISSN 3006-5496, Vol. 2, No. 3, 23 pages.

https://doi.org/10.62056/abksa6qiu

Check for updates

On the Possibility of Malicious Obfuscation

Shalini Banerjee[1] and Steven D. Galbraith[2]

[1] KASTEL SRL, Karlsruhe Institute of Technology, Germany
[2] University of Auckland, New Zealand

Abstract. We study a new variant of *malicious obfuscation*, where untrusted third-party obfuscation tools/platforms covertly insert master backdoor in software programs. We show that such malicious obfuscation could be hard to identify by the software developer who knows the original unobfuscated program. We demonstrate both undetectable and detectable malicious obfuscators for a number of obfuscation schemes in the theoretical literature, in particular conjunction obfuscation, compute-and-compare obfuscation, hamming-distance obfuscation, and point-function obfuscation.

Keywords: Program obfuscation · malicious obfuscation · third-party obfuscation

1 Introduction

Program obfuscation is used by software developers to protect intellectual property in programs [Col18]. Rather than doing the obfuscation themselves, most developers use obfuscation tools from third-parties, such as Tigress [Col23] or obfuscation-as-a-service providers [Int21]. This leads to the theoretical possibility of malicious obfuscators that cause the obfuscated program to have some undesired properties. (We stress that we are *not* implying any of the current tools are malicious; this is discussed in the Tigress FAQ [Col23].) On the mild end of the spectrum, the malicious obfuscator may result in the program having errors that were not present in the original source. On the more severe end, a malicious obfuscator could insert malware into the honest program (using it as a Trojan Horse) or could insert a master backdoor that allows access to the program when running on user devices. In fact, malicious obfuscation tools such as python-based Pyobfgood [For23] are known to embed backdoor and data-stealing routines during obfuscation.

This issue has received relatively little attention in the past. Most literature on obfuscation treats the obfuscator as honest, presumably based on the assumption that the obfuscator is controlled by the original software developer. But this does not match the reality of software development, which outsources the obfuscation service to third parties.

Canetti and Varia [CV09] introduced the notion of *verifiable obfuscation*, where they consider an end user who does not trust a developer distributing obfuscated version of a program, and hence wants to verify some predicate(s) of the program. The obfuscator itself is still considered honest in their work. Badrinarayanan, Goyal, Jain and Sahai [BGJS16] consider a similar scenario with an untrusted distributor of obfuscated programs, and aim to provide assurance to the end user running the protocol, by allowing the user to check a specific predicate on the program. Canetti, Chakraborty, Khurana, Kumar, Poburinnaya and Prabhakaran [CCK$^+$22] use a perfectly binding commitment that attests some property of the unobfuscated circuit.

E-mail: shalini.banerjee@kit.edu (Shalini Banerjee), s.galbraith@auckland.ac.nz (Steven D. Galbraith)

Received: 2025-06-30 Accepted: 2025-09-02

Given that the original developer knows the intended behaviour of the program, one might think that he could easily check the obfuscated program for malicious behaviour by simply running the program on some chosen test inputs. For example, this would detect the cheating in the password example used by Canetti and Varia [CV09]. However, the task of checking correctness of a program can often be much more complex. Also, as we will show in our examples, there are situations where a malicious obfuscator can introduce a secret hard-to-guess master password that unlocks every program, and *there is no way for the original developer to be able to guess such a password.* Hence, black-box testing the obfuscated code, while a good idea in practice, does not provide a solution to the problem.

The purpose of this paper is to initiate a study of “malicious obfuscation" *from the point of view of a software developer using an untrusted implementation of an obfuscation tool.* Our main conceptual contribution is to explain that a formal definition of malicious obfuscation can be built using the existing notions of correctness. Our main technical contribution is to demonstrate undetectable malicious obfuscators for a number of obfuscation schemes in the theoretical literature, in particular conjunction obfuscation and compute-and-compare obfuscation. Additionally, we give examples of obfuscation schemes that cannot be leveraged to introduce malicious functionality. While the concept of malicious obfuscation has been discussed by many authors, we believe *we are the first to point out undetectable malicious obfuscators for schemes in the literature.*

On the verifiability of obfuscation. While verifiability in obfuscation has been explored by [CV09, CCK$^+$22] in the context where an end-user, who *does not know the original unobfuscated program*, receives an obfuscated program and verifies some predicate(s) of the program, we consider an easier case where a software developer, who *knows the original unobfuscated program* and uses an untrusted obfuscation tool/platform (provided by a third-party), verifies whether the obfuscation procedure has been honest or malicious. One general approach to verify the correctness of obfuscation procedure (in our context) is that the obfuscator outputs a proof that can be checked by a verifier. If the proof is accepted, then the verifier can be sure that the obfuscated program is correct.

There are several ways to do this, including using non-interactive zero knowledge proofs. If we consider a model where the verifier knows how the obfuscation algorithm works (in addition to the original unobfuscated program), then zero knowledge is not necessary. A trivial non-interactive proof could be as follows: provide the random coins used by the obfuscator. The verifier simply re-computes the obfuscation using the same random coins, and checks if the output is the same. However, this approach is not very efficient as it requires the verifier to re-do the obfuscation. One can trade-off efficiency and security by just checking a random subset of the steps, to get some assurance that the obfuscation is honest.

We consider the development of efficient methods for checking the correctness of obfuscation from the point of view of a software developer using untrusted obfuscation tool/platform as an interesting open problem for future follow-up work. The main finding of our paper is that malicious obfuscation can be performed via dishonestly computing values that are supposed to be random. So, our paper shows that if one wants to use ZK proofs to certify that an obfuscator has been run correctly, then this has to go right down to the level of proving that the pseudorandom numbers used by the obfuscator are generated correctly.

1.1 Main Contributions

We now summarize our main contributions:

- We demonstrate undetectable malicious obfuscators for several well-known theoretical obfuscation proposals with strong security guarantees. In particular, we show malicious obfuscators for the conjunction obfuscators by Bishop, Kowalczyk, Malkin, Pastro, Raykova and Shi [BKM+18], and Bartesuk, Lepoint, Ma and Zhandry [BLMZ19]. We also show malicious obfuscators for the compute-and-compare construction by Goyal, Koppula, and Waters [GKW17], Wichs and Zirdelis [WZ17], and Goyal, Koppula, Vusirikala and Waters [GKVW20]. We prove that our malicious obfuscators are indistinguishable from honest ones, even with respect to an auditor who knows the original un-obfuscated program.

We argue that our attacks are meaningful, since developers may use third party obfuscation tools/platforms. A key observation is that our attacks work by replacing values that are supposed to be random by carefully chosen values. It means that any preventative measure (e.g., giving succinct proofs of correct obfuscation) will not be able to neglect the process of random number generation.

- We show existence of obfuscation schemes that cannot be leveraged to inject malicious functionality. More specifically, we show that the proposed malicious obfuscation notion does not apply to the hamming-distance obfuscator by Galbraith and Zobernig [GZ19], and the point-function obfuscator by Bartusek, Ma and Zhandry [BMZ19].

2 Notions of Obfuscation

In this section, we present background definitions for obfuscation from the literature, and introduce the definitional framework of malicious obfuscators. We follow the foundational work on obfuscation by Barak *et al.* [BGI+12], and use the circuit model for programs, although in the main body of the paper the programs will be written in pseudo-code.

Malicious obfuscation is the insertion of un-desired functionality or behaviour into a program. Our main conceptual contribution, which is not deep, is that *malicious obfuscation is therefore a violation of correctness.* Barak *et al.* [BGI+12] first consider *perfect* correctness (also called functionality preserving), where the obfuscated program $\widetilde{C}$ computes the exact same function as C. However, perfect correctness is hard to obtain in some settings, and so weaker variants of correctness have been considered, and we will review them in the case of circuits in Definition 1. Section 4.1 of [BGI+12] mentions approximate correctness, but does not give a formal definition. In fact, there are several different versions of approximate correctness in the literature, for example see [GR07, BR17, BKM+18, HMLS07]. We give two formulations of approximate correctness in Definition 1.

Definition 1 (Distributional Virtual Black-Box Obfuscator (DVBB) [BBC+14] [BGI+12]**).** *Let $\lambda \in \mathbb{N}$ be the security parameter. Let $\mathcal{C} = \{\mathcal{C}_\lambda\}$ be a family of polynomial-sized programs parameterized by inputs of length $n(\lambda)$, and let $\mathcal{D} = \{\mathcal{D}_\lambda\}$ be the class of distribution ensembles, where $\mathcal{D}_\lambda$ is a distribution over $\mathcal{C}_\lambda$. A PPT algorithm $\mathcal{O}$ with correctness ϕ_i is an obfuscator for the family $\mathcal{C}$ and the distribution $\mathcal{D}$, if it satisfies the following conditions:*

- *Correctness:*
  - ϕ_1 : *(Perfect correctness) For every $\lambda \in \mathbb{N}$ and every $C \in \mathcal{C}_\lambda$ and every possible output $\tilde{C} = \mathcal{O}(C)$,*

    $$\forall x \in \{0,1\}^{n(\lambda)} : \ \tilde{C}(x) = C(x)$$

    This is formalized in [BGI+12].
  - ϕ_2 : *(Approximate correctness) There is a negligible function $\mu(\lambda)$ such that, for every $\lambda \in \mathbb{N}$ and every $C \in \mathcal{C}_\lambda$:*

    $$\Pr_{\mathcal{O}} \left[\forall x \in \{0,1\}^{n(\lambda)} : \ \mathcal{O}(C)(x) = C(x) \right] > 1 - \mu(\lambda)$$

where the probability is over the coin tosses of $\mathcal{O}$ in computing $\tilde{C} = \mathcal{O}(C)$. This is formulated in Definition 3.1 of [WZ17].

- ϕ_3 : *(Approximate correctness) There is a negligible function $\mu(\lambda)$ such that, for every $\lambda \in \mathbb{N}$, for every $C \in \mathcal{C}_\lambda$, and for every $x \in \{0,1\}^{n(\lambda)}$ to C:*

$$\underset{\mathcal{O}}{Pr}\ [\ \mathcal{O}(C)(x) = C(x)\ ] > 1 - \mu(\lambda)$$

where the probability is over the coin tosses of $\mathcal{O}$. This is called "weak functionality preservation" in [BLMZ19].

- *Polynomial Slowdown : For every $\lambda \in \mathbb{N}$ and for every $C \in \mathcal{C}_\lambda$, there exists a polynomial q such that the running time of $\tilde{C} = \mathcal{O}(C)$ is bounded by $q\left(|C|\right)$, where $|C|$ denotes the size of the program.*

- *Virtual Black-box : For every (non-uniform) polynomial size adversary $\mathcal{A}$, there exists a (non-uniform) polynomial size simulator $\mathcal{S}$ with oracle access to C, such that for every distribution $D \in \mathcal{D}_\lambda$:*

$$\left| \underset{C \leftarrow \mathcal{D}_\lambda, \mathcal{O}, \mathcal{A}}{Pr}[\mathcal{A}(\mathcal{O}(C)) = 1] - \underset{C \leftarrow \mathcal{D}_\lambda, \mathcal{S}}{Pr}[\mathcal{S}^C(1^\lambda) = 1] \right| \leq \mu(\lambda)$$

where $\mu(\lambda)$ is a negligible function.

We note that the constructions we study in this paper mainly follow ϕ_2 and ϕ_3 correctness, and belong to the family of programs with the property that for a fixed input, a random $C \in \mathcal{C}_\lambda$ evaluates to 0 with overwhelming probability (evasive programs).

2.1 Defining Malicious Obfuscators

Our formalism of malicious obfuscators is that the output program does not have the same correctness guarantee as the honestly obfuscated program. For example, suppose the malicious obfuscator inserts a master backdoor y that is accepted by every program it obfuscates. Then correctness ϕ_1 and ϕ_2 are not possible (a program is never correct on all inputs) and correctness ϕ_3 is also not possible (for the specific input y, we have $C(y) = 0$ but $\mathcal{O}(C)(y) = 1$).

In addition, we require that a polynomial distinguisher cannot detect that $\tilde{C}$ is maliciously generated, either by inspecting the source code of $\tilde{C}$, running it on chosen inputs, or both.

In reality, an obfuscation tool is used by many users to obfuscate many programs. Hence it is necessary for a malicious obfuscator to be undetectable even when used to obfuscate many programs. Therefore our security definition allows the distinguisher to receive obfuscations of polynomially many chosen programs C, including repeated obfuscations of the same program. We do this by providing oracle access to the obfuscator. We also consider the case where the malicious obfuscator may be introducing a master backdoor that is the same for every execution. This is modeled in our formalism as a fixed auxiliary input aux that is used for all executions.

We stress that our definition holds for *any* aux, which we demonstrate later in the concrete realizations of our notion. Given the malicious obfuscator could be a software tool that is sold by an obfuscation company and executed locally and offline by a software developer to obfuscate many programs, it does not make sense to choose aux adaptively as the adversary never sees the input. Moreover, aux should be hard to determine by the distinguisher, and hence we require aux to be sampled randomly.

Definition 2 (**Malicious Obfuscation**). *Let λ, n satisfy the conditions as given in Definition 1. For any family of programs $\mathcal{C}$ and distribution class $\mathcal{D}$ over $\mathcal{C}$, let $\mathcal{O}$ be an* *obfuscator that satisfies the conditions given in Definition 1 with correctness ϕ_i. Then a* malicious obfuscator *for the family $\mathcal{C}$ and distribution $\mathcal{D}$ is a PPT algorithm $\mathcal{A}$ that takes an auxiliary input $\mathsf{aux} \in \{0,1\}^\lambda$, such that:*

- *(Correctness violation) : For any choice of aux, $\mathcal{A}(1^\lambda, \cdot, \mathsf{aux})$ does not satisfy ϕ_i.*
- *(Indistinguishability) : There exists a negligible function $\mu(\lambda)$, such that for every PPT distinguisher $\mathcal{B}$ that has oracle access to an obfuscator (so can adaptively ask for obfuscations of polynomially many adaptively chosen $C \in \mathcal{C}_\lambda$)*

$$\left| \Pr_{\mathsf{aux},\mathcal{B},\mathcal{A}} [\, \mathcal{B}^{\mathcal{A}(1^\lambda,\cdot,\mathsf{aux})} = 1] - \Pr_{\mathcal{B},\mathcal{O}} [\, \mathcal{B}^{\mathcal{O}(1^\lambda,\cdot)} = 1] \right| \leq \mu(\lambda)$$

  where the first probability is taken over the choice of aux and the coin tosses of $\mathcal{B}$, $\mathcal{A}$, and the second probability is taken over the coin tosses of $\mathcal{B}$, $\mathcal{O}$.

The indistinguishability game is that a distinguisher can request and receive obfuscations of C. We write $\mathcal{B}^{\mathcal{O}(\cdot)}$ to indicate that $\mathcal{B}$ has access to an oracle that can be queried on any C (but with respect to fixed aux in the malicious case). The string aux represents randomly generated secret data that is known to the malicious obfuscator, such as the value of a master backdoor. The distinguisher has to decide if it is interacting with the honest obfuscator or a malicious one.

3 Malicious Obfuscators for Conjunctions

We now show that malicious obfuscators exist for schemes in the literature, that are indistinguishable from honest obfuscators. In this section we focus on conjunctions, which are also called pattern matching with wildcards.

3.1 Reviewing the [BKM+18] Construction

We now review the construction by Bishop *et al.* [BKM+18] for obfuscating conjunctions (alternatively called pattern matching with wildcards), and design a malicious obfuscator, seemingly identical to the honest obfuscation instance. We first recall the definition of conjunctions.

Definition 3 (**Conjunctions**). *Let $n \in \mathbb{N}$ and let $\mathsf{pat} \in \{0,1,\star\}^n$ be a pattern, where $\star$ is a wildcard character. Let $W = \{i : \mathsf{pat}_i = \star\}$ be the set of wildcard positions in pat, and let $w = |W|$. A conjunction function $C : \{0,1\}^n \to \{0,1\}$, $x \mapsto C(x)$ on an input $x \in \{0,1\}^n$ is defined as*

$$C(x) = \begin{cases} 1\,, & \text{if } \forall i \text{ such that } \mathsf{pat}_i \neq \star \wedge x_i = \mathsf{pat}_i \\ 0\,, & \text{otherwise.} \end{cases}$$

Bishop *et al.* designed an efficient DVBB obfuscator for conjunction functions using Lagrange interpolation. Their security goal roughly states that a PPT adversary cannot distinguish the obfuscation of C from obfuscation of a function that always outputs 0. The construction satisfies "approximate" functionality preservation (ϕ_3 in Definition 1) for an ensemble of uniform distributions. The scheme relies on the difficulty of the discrete logarithm problem in a group of size q, and the security proof takes place in the generic group model. Hence we need $q > 2^{2\lambda}$ where λ is the security parameter.

The high-level overview of the obfuscation is as follows: to obfuscate $\mathsf{pat} \in \{0,1,\star\}^n$ that has w wildcards, define polynomial $F(t) = \sum_{k=1}^{n-1} a_k t^k \in \mathbb{F}_q[t]$ with $F(0) = 0$. The coefficients $a_1, \ldots, a_{n-1}$ are sampled uniformly random in $\mathbb{F}_q$, where q is exponential in λ. If the ith bit of the pattern is j, where $j \in \{0,1\}$ or if $\mathsf{pat}_i = \star$, then evaluate the

Algorithm 1 Obfuscator $\mathcal{O}_{Con}(1^\lambda, C)$

```
1:  Sample large prime $q > 2^{2\lambda}$; Select $\mathbb{G} = \langle g \rangle$ of order $q$
2:  Sample $(a_1, \dots, a_{n-1}) \xleftarrow{\$} \mathbb{F}_q$
3:  Let $F(t) = a_1 t^1 + \cdots + a_{n-1} t^{n-1}$
4:  for $i = 1$ to $n$ do
5:      for $j = 0$ to $1$ do
6:          if $(\mathsf{pat}_i == \star \vee \mathsf{pat}_i == j)$ then
7:              $h_{i,j} \leftarrow g^{F(2i+j)}$
8:          else
9:              $h_{i,j} \xleftarrow{\$} \mathbb{F}_q$
10:         end if
11:     end for
12: end for
13: return $v = (g^{h_{i,j}})_{i \in [n], j \in \{0,1\}}$
```

polynomial at $2i + j$, otherwise sample a uniformly random element from $\mathbb{F}_q$. The final step is to publish the $2n$ field elements in the exponent of the group $\mathbb{G} = \langle g \rangle$ of order q. The formal description is given in Algorithm 1, where the input C may be viewed as a circuit that computes the conjunction function, or as a description of the pattern (in either case, it is assumed that it is easy to determine pat from C).

Interpolating the polynomial in the exponent with n Lagrange coefficients corresponding to a correct input $x \in \{0,1\}^n$ gives g^0, and the evaluator correctly accepts the input. For an input that does not match the pattern, a uniformly random group element is returned by the algorithm, and the evaluator correctly rejects the input with overwhelming probability.

3.2 Malicious Obfuscator for [BKM+18]

As is clear from Definition 3, a conjunction function C for a pattern $\mathsf{pat} \in \{0, 1, \star\}^n$ with w wildcard characters, defines 2^w accepting inputs. Our goal is to design a malicious obfuscator that allows a certain input string y that does not match the pattern.

The input y should be fixed and independent of the pattern being obfuscated (so that y can be a single master backdoor that works for all instances). Furthermore, we require that any poly-time distinguisher $\mathcal{B}$ with a priori knowledge on pat cannot distinguish between honest and purported obfuscation instances.

The malicious obfuscator must accept the 2^w inputs strings that correctly match the pattern. This means that for every $\mathsf{pat}_i = \star$ or $\mathsf{pat}_i = j$, the elements $h_{i,j}$ should be correctly structured. Given we require the obfuscator to accept bad inputs, a naïve solution would be to output $h_{i,j} \leftarrow g^{F(2i+j)}$ for every $i \in [n]$ and $j \in \{0,1\}$, as the obfuscator would then accept all strings of length n. Slightly more cleverly, one could add wildcard positions only in those places where y_i does not match the pattern (namely, we are setting $\mathsf{pat}_i = \star$ for all i such that $\mathsf{pat}_i \in \{0,1\}$ but $\mathsf{pat}_i \neq y_i$. Certainly such an obfuscated program would accept the master backdoor y, but a PPT distinguisher who knows pat can simply flip the $n - w$ non-wildcard bits one by one, and check if any of the inputs are accepted in $\mathrm{O}(n)$ time. Hence it is easy to detect this malicious behaviour. Thus it remains to construct an un-detectable malicious obfuscator around the apparent constraints.

The basic idea is to choose the "random" values in Algorithm 1 in a structured way so that the polynomial interpolation also works for the input y. First, since y is fixed and independent of C, it might happen that $C(y) = 1$ anyway. In this case there is no malicious behaviour and the original obfuscation works. Similarly, if y matches the pattern except for one non-wildcard position (call it k) then the malicious obfuscator is exactly the same as an obfuscation of the pattern where location k is now a wildcard. In this case it is impossible to have an obfuscated program that accepts y but that can't be detected by a distinguisher who checks the wildcard positions.

Hence in reality we focus on the case where $C(y) = 0$ and where y does not match the pattern in at least two positions (this is the case with overwhelming probability for evasive conjunctions). Let $\mathcal{L}_y$ be the set of indices in y that do not satisfy the pattern. Note that such a string would be correctly rejected with overwhelming probability by the [BKM+18] construction. We require the malicious obfuscator to agree with the honest obfuscator except for the input string y, which it allows with noticeable advantage.

To achieve this, we construct the random degree $n-1$ polynomial $F(t)$ following Algorithm 1, *while we define another polynomial* $H(t) \in \mathbb{F}_q[t]$, such that $H(0) = 0$, and H evaluates to the same value as F at the positions where y_i satisfies pat_i. Such a polynomial exists and can be chosen to be not equal to F if there are at least two positions where y does not match the pattern, as two distinct degree $n-1$ polynomials may intersect at $n-1$ points. If y matches the pattern except at one position, then we have $H = F$, and that position behaves as a wildcard. The obfuscation follows Algorithm 1, except that certain random choices are now defined using H, so that y is accepted. This ensures that the obfuscated program always accepts all inputs satisfying the pattern, as well as the input y. Other inputs are rejected with overwhelming probability. A distinguisher who does not know y is unable to check whether the allegedly random group elements have instead been generated using the polynomial H. The formal description is given in Algorithm 2.

Lemma 1. *Let $\lambda \in \mathbb{N}$ be the security parameter. Let $\mathcal{C} = \{\mathcal{C}_\lambda\}$ be the family of polynomial-sized conjunction programs parameterized by inputs of length $n = n(\lambda)$, and let $\mathcal{D} = \{\mathcal{D}_\lambda\}$ be a class of distribution ensembles, where $\mathcal{D}_\lambda$ is a distribution over evasive conjunctions in $\mathcal{C}_\lambda$. Then algorithm $\mathcal{A}_{Con}$ (Algorithm 2) violates ϕ_3. Precisely, for every $y \in \{0,1\}^n$, there exists a $C \in \mathcal{D}_\lambda$ and $x \in \{0,1\}^n$ such that*

$$Pr[\, \mathcal{A}_{Con}(1^\lambda, C, y)(x) \neq C(x)\,] > 1 - \mu(\lambda)$$

for some negligible function μ.

Proof. Let $y \in \{0,1\}^n$. Since $\mathcal{D}_\lambda$ is a distribution on evasive conjunction functions, there exists some C such that $C(y) = 0$ (indeed, one can efficiently find such a C in expected constant time, since $C(y) = 0$ with overwhelming probability over the choice of C). Now, by construction, $\mathcal{A}_{Con}(1^\lambda, C, y)(y) = 1$ with probability 1. So taking $x = y$, the result follows.

□

One can prove that the scheme is indistinguishable in the generic group model, but due to lack of space we omit the security proof and instead give the proof details for the more efficient construction from [BLMZ19]. The indistinguishability shows that the malicious obfuscator is also distributional VBB in the generic group model.

3.3 Reviewing the [BLMZ19] Construction

We now present the dual version of conjunction obfuscation by Bartesuk, Lepoint, Ma and Zhandry [BLMZ19], which is more efficient than the [BKM+18] construction. The dual scheme takes into account evasive conjunctions with patterns of length n, and achieves distributional virtual black box security for $n - \omega(\log n)$ wildcards in the generic group model with $n+1$ group elements, rather than $2n$.

Definition 4. *Let* $\mathbf{B}$ *be the* $(n+1) \times 2n$ *dimensional matrix* with (i,j)-entry j^i.

Then matrix $\mathbf{B}$ has the property that any of its $n+1$ columns form a full rank matrix.

To encode a pattern $\mathsf{pat} \in \{0,1,\star\}^n$, compute a $2n$ dimensional *error vector* $\mathbf{e}$ structured as follows: if the ith bit of the pattern is b, then $\mathbf{e}_{2i-b} = 0$, otherwise $\mathbf{e}_{2i-(1-b)}$ is sampled

Algorithm 2 Malicious Obfuscator $\mathcal{A}_{Con}(1^\lambda, C, y \in \{0,1\}^n)$

```
 1: Sample large prime $q > 2^{2\lambda}$; Select $\mathbb{G} = \langle g \rangle$ of order $q$
 2: if $(C(y) == 1)$ then
 3:   return $\mathcal{O}_{Con}(1^\lambda, C)$
 4: else
 5:   Sample $(a_1, \dots, a_{n-1}) \xleftarrow{\$} \mathbb{F}_q$
 6:   $F(t) = a_1 t^1 + \cdots + a_{n-1} t^{n-1}$
 7:   $\mathcal{L}_y := \{1 \le k \le n : y_k \ne \mathsf{pat}_k\}$
 8:   Sample random degree $n-1$ polynomial $H(t) \in \mathbb{F}_q[t]$ s.t. $H(0) = 0$ and $H(2k+y_k) =$
      $F(2k + y_k)$ for all $k \notin \mathcal{L}_y$
 9:   for $i = 1$ to $n$ do
10:     for $j = 0$ to $1$ do
11:       if $(\mathsf{pat}_i == \star \vee \mathsf{pat}_i == j)$ then
12:         $h_{i,j} \leftarrow g^{F(2i+j)}$
13:       else
14:         if $i \in \mathcal{L}_y$ then
15:           $h_{i,j} \leftarrow g^{H(2i+y_i)}$
16:         else
17:           $h_{i,j} \xleftarrow{\$} \mathbb{F}_q$
18:         end if
19:       end if
20:     end for
21:   end for
22: end if
23: return $v^* = (g^{h_{i,j}})_{i \in [n], j \in \{0,1\}}$
```

randomly from $\mathbb{Z}_q$. If $\mathsf{pat}_i = \star$, then $\mathbf{e}_{2i-1} = \mathbf{e}_{2i} = 0$. The obfuscator outputs the encoding of the vector $\mathbf{Be}$ in the exponent of the group $\mathbb{G} = \langle g \rangle$, as $g^{\mathbf{Be}} \in \mathbb{G}^{n+1}$.

On input string $x \in \{0,1\}^n$, the evaluation procedure solves for a vector $\mathbf{t}$, such that $\mathbf{tB} = 0$ at positions $2i - (1 - x_i)$, for every $i \in [n]$. Finally, x is accepted if $\mathbf{tBe} = 0$, which is tested by computing in the group.

The correctness of the scheme is proved by Bartesuk, Lepoint, Ma and Zhandry [BLMZ19]. They prove DVBB security (for certain parameter ranges and for uniformly sampled patterns) in the generic group model. Bartusek *et al.* [BLMZ19] claim to achieve *correctness* ϕ_3 (see Definition 1).

3.4 Malicious Obfuscator for [BLMZ19]

To construct an obfuscated program that accepts the bad input $y \in \{0,1\}^n$ with noticeable advantage, we again replace random values with specially chosen values. We assume y does not match pat (i.e. $C(y) = 0$ in Definition 3) otherwise we just have to return an honest obfuscation. Hence we may assume that y does not match the pattern in at least one entry. Since any $n+1$ columns of $\mathbf{B}$ are linearly independent, $\mathbf{B}_y \in \mathbb{Z}_q^{(n+1)\times n}$ will have rank n and there is a unique one-dimensional space of vectors $\mathbf{t}_y \in \mathbb{Z}_q^{1\times(n+1)}$, such that $\mathbf{t}_y \mathbf{B}_y = 0$ (by the rank-nullity theorem). Since $\mathbf{B}_y$ differs from $\mathbf{B}_x$ for every x that satisfies the pattern, the vector $\mathbf{t}_y$ differs from the vector $\mathbf{t}$ for any honest x.

Recall that the $2n$ dimensional vector $\mathbf{e}$ has $n + w$ zero entries by construction. To design a purported error vector $\mathbf{e}^*$ that works with both y and the correct inputs, we fix the $n+w$ positions with zero entries (corresponding to the honest obfuscation). Computing $\mathbf{e}^*$ is done by finding a non-zero vector in the $(2n-1)$-dimensional subspace orthogonal to

Algorithm 3 Obfuscator $\mathcal{O}_{Dual}(1^\lambda, C)$

```
1: Sample large prime $q > 2^{2\lambda}$; Select $\mathbb{G} = \langle g \rangle$ of order $q$
2: Let $\mathbf{B} \in \mathbb{Z}_q^{(n+1)\times 2n}$ as in Definition 4
3: Initialize error vector $\mathbf{e} \leftarrow \mathbb{Z}_q^{2n\times 1}$
4: for $i = 1$ to $n$ do
5:   if $(\mathsf{pat}_i == \star)$ then
6:     $\mathbf{e}_{2i-1} = \mathbf{e}_{2i} = 0$
7:   end if
8:   if $(\mathsf{pat}_i == b)$ then
9:     $\mathbf{e}_{2i-b} = 0$ ; $\mathbf{e}_{2i-(1-b)} \xleftarrow{\$} \mathbb{Z}_q$
10:  end if
11: end for
12: return $\mathbf{B}, \mathbf{v} = g^{\mathbf{Be}}$
```

$\mathbf{tB}$ that also has the correctly structured zero entries. Let $\mathbf{U} \in \mathbb{Z}_q^{2n\times 1}$ be the subspace of vectors with basis $\{\mathbf{u}_{2i-(1-b)} : \mathsf{pat}_i = b\}$. Let $\mathbf{U}' = \{\mathbf{w} : \mathbf{w} \in \mathbf{U} \wedge \mathbf{t}_y\mathbf{B}\mathbf{w} = 0\}$. Since $n + w$ are fixed zero entries, the dimension of $\mathbf{U}'$ is $n - w - 1$. Thus, for an input y that does not match pat, if the obfuscator selects a vector $\mathbf{e}^*$ from $\mathbf{U}'$ and publishes $\mathbf{Be}^*$, the evaluation algorithm will accept y, as $\mathbf{t}_y\mathbf{Be}^* = 0$. We specify the procedure formally in Algorithm 4.

Algorithm 4 Malicious Obfuscator $\mathcal{A}_{Dual}(1^\lambda, C, y \in \{0,1\}^n)$

```
1: Sample large prime $q > 2^{2\lambda}$; Select $\mathbb{G} = \langle g \rangle$ of order $q$
2: Let $\mathbf{B} \in \mathbb{Z}_q^{(n+1)\times 2n}$ as in Definition 4
3: if $(C(y) == 1)$ then
4:   return $\mathcal{O}_{Dual}(1^\lambda, C)$
5: else
6:   Define $\mathbf{B}_y \in \mathbb{Z}_q^{(n+1)\times n}$, where column $j$ is set to $(\mathbf{B}_y)_j = \mathbf{B}_{2j-y_j}$
7:   Solve for non-zero vector $\mathbf{t}_y \in \mathbb{Z}_q^{1\times(n+1)}$ such that $\mathbf{t}_y\mathbf{B}_y = 0$
8:   Compute $\mathbf{U} \in \mathbb{Z}_q^{2n\times 1}$ with the basis $\{\mathbf{u}_{2i-(1-b)} : \mathsf{pat}_i = b\}$
9:   Compute $\mathbf{U}' = \{\mathbf{w} : \mathbf{w} \in \mathbf{U} \wedge \mathbf{t}_y\mathbf{B}\mathbf{w} = 0\}$
10:  Sample $\mathbf{e}^* \xleftarrow{\$} \mathbf{U}'$
11:  return $\mathbf{B}, \mathbf{v}^* = g^{\mathbf{Be}^*}$
12: end if
```

Lemma 2. *Let $\lambda \in \mathbb{N}$ be the security parameter. Let $\mathcal{C} = \{\mathcal{C}_\lambda\}$ be the family of conjunction programs parameterized by inputs of length $n = n(\lambda)$, and let $\mathcal{D} = \{\mathcal{D}_\lambda\}$ be a class of distribution ensembles, where $\mathcal{D}_\lambda$ is a distribution over evasive conjunction programs in $\mathcal{C}_\lambda$. Then Algorithm 4 violates ϕ_3. Precisely, for each $y \in \{0,1\}^n$ there exists a $C \in \mathcal{D}_\lambda$ and an $x \in \{0,1\}^{n(\lambda)}$ such that*

$$Pr[\,\mathcal{A}_{Dual}(C, y)(x) \neq C(x)\,] > 1 - \mu(\lambda)$$

for some negligible function $\mu(\lambda)$.

Proof. It is proved by Bartusek *et al.* [BLMZ19] that the dual scheme satisfies ϕ_3. Hence the following holds for every C and every $x \in \{0,1\}^n$:

$$\Pr_{\mathcal{O}_{Dual}, C}[\,\mathcal{O}_{Dual}(1^\lambda, C)(x) = C(x)\,] > 1 - \mu(\lambda)$$

To show that a malicious obfuscator does not satisfy ϕ_3 it suffices to exhibit a single C and x such that $\mathcal{A}_{Dual}(C,y)(x) \neq C(x)$ with high probability.

Let $y \in \{0,1\}^n$. Since $\mathcal{D}_\lambda$ is a distribution on evasive conjunction functions, there exists some C such that $C(y) = 0$. Now, by construction, $\mathcal{A}_{Dual}(1^\lambda, C, y)(y) = 1$ with probability 1. So taking $x = y$, the result follows.

□

3.5 Indistinguishability of our Malicious Obfuscator

We now prove computational indistinguishability of the malicious and honest obfuscators (specified in Algorithms 3 and 4) in the generic group model.

The generic group model (GGM) is an abstract model of computation where a generic adversary can access the group structure through oracle calls but cannot exploit the representation of the group elements. The adversary is initialized with "handles" that represent the GGM group elements. This model is used for the security proofs in [BKM+18, BLMZ19], so it is natural to use it here.

We use the same formulation of generic group as in Bartesuk, Lepoint, Ma and Zhandry [BLMZ19]. Specifically, the two oracle calls available are: (i) $\mathsf{sub}(\sigma_1,\sigma_2)$, which computes the handle corresponding to the group element $x - y$, where σ_1 is a handle for x and σ_2 is a handle for y; (ii) $\mathsf{isZero}(\sigma)$, which returns true if and only if the handle corresponds to the identity element of the group.

The fact that our malicious obfuscator is indistinguishable from the honest obfuscator in the generic group model implies the distributional VBB security of the malicious obfuscator under the same conditions as required in [BLMZ19].

We consider a distinguisher $\mathcal{B}$ that can request polynomially many obfuscated programs and determine whether it is interacting with an honest or malicious obfuscator. We require $\mathcal{B}$ to be a generic algorithm, and do not give it direct access to the group. Instead, group elements are replaced by abstract "handles", and the algorithm $\mathcal{B}$ is required to make oracle queries to compute group operations (where the group is viewed as an additive group).

Theorem 1. *Let $\lambda \in \mathbb{N}$ be the security parameter and let n, w be polynomials in λ with $w = n - \omega(\log n)$. Let $\mathbb{G}$ be a group of prime order $q > 2^{2\lambda}$.*

Then for all PPT distinguishers $\mathcal{B}$ in the generic group model, there exists a negligible function $\mu(\lambda)$, such that for all $\lambda \in \mathbb{N}$, the following holds:

$$\left| \Pr_{y,\mathcal{B},\mathcal{A}_{Dual}} [\,\mathcal{B}^{\mathcal{A}_{Dual}(1^\lambda,\cdot,y)} = 1] - \Pr_{\mathcal{B},\mathcal{O}_{Dual}} [\,\mathcal{B}^{\mathcal{O}_{Dual}(1^\lambda,\cdot)} = 1] \right| \leq \mu(\lambda)$$

Proof. We prove the theorem by designing a simulator that plays the role of the challenger in the security game with $\mathcal{B}$ and controls the generic group oracles.

The intuition behind the proof is that the simulator "decides" whether to play as an honest or malicious obfuscator only *after* the algorithm $\mathcal{B}$ has returned its guess.

Let T be an upper bound on the number of queries made by $\mathcal{B}$ to the obfuscation oracle. For simplicity of notation we assume all obfuscation queries are with respect to the same parameters n, w and use the same prime q (group order). The general case where the groups are varying is handled using a hybrid argument. To simulate the generic group we will need to work with linear polynomials in Tn variables. So we work in the ring $R = \mathbb{Z}_q[X_i^{(t)}]$ for $1 \leq i \leq n$ and $1 \leq t \leq T$.

To begin the simulator initializes a list $\mathcal{L} \leftarrow \{\,\}$, which will keep track of the generic group queries. We set $t = 1$, as the counter for the number of obfuscation queries.

For each query to obfuscate a program C, the simulator first derives from C the pattern pat. The simulator must set up the $n+1$ handles that will be provided to $\mathcal{B}$. Handles are strings in $\{0,1\}^\tau$, where $\tau > \log_2(q)$. This is done in the $\mathcal{O}(C)$ query part of Algorithm 5.

This ensures that the group elements correspond to $\mathbf{Be}$ for some vector $\mathbf{e}$ that satisfies the requirements of the scheme. The vector $\mathbf{e}$ has length $2n$, and has $n+w$ entries fixed to zero and $n-w$ entries that are supposed to be random group elements. The idea of the simulation is to treat the $n-w$ entries as indeterminates. Hence, in the i-th execution the simulation introduces $n-w$ variables $X_1^{(t)}, \ldots, X_n^{(t)}$ (we only use $n-w$ variables, but for simplicity of notation we go all the way to n) and sets $\mathbf{e}$ to be the vector whose non-wildcard entries are variables. The simulation then returns $n+1$ random handles to $\mathcal{B}$, where each handle is associated with the polynomial that is given by the corresponding entry of $\mathbf{Be}$.

Note that, by construction, if $\mathcal{B}$ executes the obfuscated program on an input that matches the pattern, then the final isZero query will return True and the input will be accepted. On the other hand, if $\mathcal{B}$ executes the obfuscated program on an input that does not match the pattern, then the final isZero query will return False and the input will be rejected.

Algorithm 5 Oracle Handler

```
$\boxed{\mathcal{O}(C)\ \textbf{query}}$

1: $\mathbf{e}^{(t)} \in R^{2n}$ s.t. $R = \mathbb{Z}_q[X_1^{(t)}, \ldots, X_n^{(t)}]$ is a zero vector
2: for $i = 1$ to $n$
3:     if ($\mathsf{pat}_i == b$)
4:         $\mathbf{e}_{2i-(1-b)}^{(t)} \leftarrow X_i^{(t)}$
5:     end if
6: end for
7: Let $(F_1, \ldots, F_{n+1}) = \mathbf{Be}^{(t)} \in R^{n+1}$
8: Sample random handles $(\sigma_1, \ldots, \sigma_{n+1})$ from $\{0,1\}^\tau$
9: $\mathcal{L} \leftarrow \mathcal{L} \cup \{(\sigma_i, F_i)_{i\in[n+1]}\}$
10: $t \leftarrow t+1$
11 return: $(\sigma_1, \ldots, \sigma_{n+1})$

$\boxed{\mathsf{Sub}(\sigma_i, \sigma_j)\ \textbf{query}}$

1: Find $F_i, F_j \in \mathbb{Z}_q[X_1^{(1)}, \ldots, X_n^{(T)}]$ s.t. $(\sigma_i, F_i) \in \mathcal{L}$ and $(\sigma_j, F_j) \in \mathcal{L}$ (if they don't exist return $\perp$)
2: $F = F_i - F_j$
3: if $\exists\, \sigma : (\sigma, F) \in \mathcal{L}$ return $\sigma$
4: else
5: $\sigma \xleftarrow{\$} \{0,1\}^\tau$;
6: $\mathcal{L} \leftarrow \mathcal{L} \cup \{(\sigma, F)\}$
7: end if

$\boxed{\mathsf{isZero}(\sigma)\ \textbf{query}}$

1: $F \in \mathbb{Z}_q[X_1^{(1)}, \ldots, X_n^{(T)}]$ s.t. $(\sigma, F) \in \mathcal{L}$ (return $\perp$ if none exists)
2: if ($F == 0$) return True
3: else return False
4: end if
```

After polynomially many oracle queries, $\mathcal{B}$ outputs a bit b'. The simulator now generates a random bit b and proceeds as follows.

If $b = 0$ then the simulator chooses random vectors $(x_1^{(t)}, \ldots, x_n^{(t)}) \in \mathbb{Z}_q^n$ for each $1 \leq t \leq T$. We now fix the vector $\mathbf{e}^{(t)}$ by evaluating the polynomials on the point $(x_1^{(t)}, \ldots, x_n^{(t)})$. The values $\mathbf{Be}^{(t)}$ are now distributed as in the honest obfuscation. The simulation has been correct as long as all isZero queries were answered consistently with this choice of group element. If isZero answered true then the answer was correct, but some false answers (on non-zero polynomials) may have been incorrect. Let Q be the number of isZero queries, which is an upper bound on the number of non-zero polynomials F that might have had the point $(x_1^{(1)}, \ldots, x_n^{(T)})$ as a root. Note that Q is bounded by a polynomial in the security parameter. Since the non-zero polynomials F in queries to isZero are all linear, by the Schwartz–Zippel lemma and the union bound, the probability the simulation is incorrect is at most Q/q, which is negligible since $q > 2^{2\lambda}$.

Now consider the case when $b = 1$. Then the simulator chooses a malicious input $y \in \{0,1\}^n$. This represents the choice of aux in Definition 2. Since there are polynomially many chosen C, each corresponding to some evasive pattern pat, then y does not satisfy any of the patterns with overwhelming probability.

Now, we imagine choosing the vectors $\mathbf{e}^{(t)}$ as in the malicious scheme. Once again, this is the same as choosing a point $(x_1^{(t)}, \ldots, x_n^{(t)}) \in \mathbb{Z}_q^n$, but this time subject to the additional linear constraint $\mathbf{t}_y \mathbf{B}\mathbf{e}^{(t)} = 0$. We model this by defining additional linear polynomials $F_0(X_1^{(t)}, \ldots, X_n^{(t)})$ that correspond to the condition $\mathbf{t}_y \mathbf{B}\mathbf{e}^{(t)} = 0$. The simulation chooses the point $(x_1^{(t)}, \ldots, x_n^{(t)}) \in \mathbb{Z}_q^n$ uniformly at random. Again, the only things we need to be concerned about are whether $F_0(x_1^{(t)}, \ldots, x_n^{(t)}) = 0$ and whether queries to isZero were answered incorrectly. This is the same as the case $b = 0$, except there are T additional linear constraints. Hence, as in the previous case, the probability the simulation is incorrect is bounded by $(Q + T)/q$, which is negligible. Let $p = Pr_{\mathcal{B}, \mathcal{O}_{Dual}} \left[\mathcal{B}^{\mathcal{O}_{Dual}(1^\lambda, \cdot)} = 1 \right.$ be the probability that $\mathcal{B}$ outputs 1 in the honest game. Let Fail be the event that the simulation answers an isZero query incorrectly when $b = 0$. Then in the simulation we have $p = Pr[\mathcal{B} = 1 | \neg \mathsf{Fail}]$. Hence, in the simulation we have $Pr_{\mathcal{B}, \mathcal{O}_{Dual}} \left[\mathcal{B}^{\mathcal{O}_{Dual}(1^\lambda, \cdot)} = 1 \right] = p \, Pr[\neg \mathsf{Fail}] + Pr[\mathcal{B} = 1 | \mathsf{Fail}] \, Pr[\mathsf{Fail}] = p + \mu_1(\lambda)$ for some negligible function $\mu_1(\lambda)$.

Similarly, let Fail' be the event that the simulation is incorrect when $b = 1$. We have shown that $Pr[\mathsf{Fail}']$ is negligible. When event Fail' does not occur, the view of $\mathcal{B}$ is identical to the view of $\mathcal{B}$ when playing the game with $b = 0$ against an honest obfuscator, since none of the generic group queries have detected the additional linear equation that is due to the malicious choice of input y. Hence $Pr\left[\mathcal{B}^{\mathcal{A}_{Dual}} = 1 | \neg \mathsf{Fail}' \right] = p$. It follows that $Pr\left[\mathcal{B}^{\mathcal{A}_{Dual}} = 1 \right] = p + \mu_2(\lambda)$, for some negligible function $\mu_2(\lambda)$. $\square$

4 Malicious Obfuscators for Compute-and-Compare Programs

Compute-and-compare obfuscation is a very general tool that solves a wide class of obfuscation problems. In fact, almost all previous provable obfuscation schemes for evasive functions are special cases of evasive compute-and-compare programs. Solutions to compute-and-compare obfuscation have been given by Wichs and Zirdelis [WZ17], and Goyal, Koppula and Waters [GKW17] (who call it "lockable obfuscation"). The two schemes are very similar and are both based on the *learning with errors* (LWE) assumption. Goyal, Koppula, Vusirikala and Waters [GKVW20] design perfectly correct obfuscators for the [GKW17, WZ17] schemes. Our techniques apply to all the three schemes, but we only present the details for the obfuscation scheme by Wichs and Zirdelis [WZ17], as this gives the main idea for the constructions.

We show that malicious obfuscators exist for compute-and-compare obfuscation constructions. This is a particularly important class, since it is not necessarily easy to reverse-engineer a compute-and-compare function even when given the original program. For example, if the function computes a cryptographic hash H then one can obfuscate the program "Does $H(x) = h$?" without knowing an accepting input. Our malicious obfuscator can sample its own secret input x_0 and compute $h_0 = H(x_0)$ and ensure that the obfuscated program accepts inputs that evaluate to h or h_0, giving the malicious obfuscator a master backdoor even though it would otherwise have been hard to find an input when given the original (unobfuscated) program.

Definition 5 (Compute-and-compare Programs). *Let ℓ_{in}, $\ell_{out} \in \mathbb{N}$. Let $f : \{0,1\}^{\ell_{in}} \to \{0,1\}^{\ell_{out}}$ be a polynomial-time computable function and let $\alpha \in \{0,1\}^{\ell_{out}}$ be a target value. A compute-and-compare program on input $x \in \{0,1\}^{\ell_{in}}$ is defined as*

$$CC[f, \alpha](x) = \begin{cases} 1\,, & \text{if } f(x) = \alpha \\ 0\,, & \text{otherwise.} \end{cases}$$

We focus on the case of evasive compute-and-compare programs, meaning that if one samples a random y and sets $\alpha = f(y)$, then there is a negligible probability that other random inputs x satisfy $f(x) = \alpha$. Indeed, Wichs and Zirdelis [WZ17] require α to have high pseudo-entropy in the security parameter λ and the branching program length, which requires ℓ_{out} to be sufficiently large.

4.1 Reviewing the [WZ17] Construction

We now review the compute-and-compare obfuscator by Wichs and Zirdelis [WZ17] that achieves DVBB security under the *learning with errors* (LWE) assumption. Their scheme employs GGH15 encodings by Gentry, Gorbunov and Halevi [GGH15] in a restricted setting. We state the LWE problem by Regev [Reg09].

Definition 6 (**Decisional LWE (DLWE)**). *Let $q \in \mathbb{N}$ be a large prime and let n, $m \in \mathbb{N}$. Let χ be a noise distribution over $\mathbb{Z}_q$. The (n, q, χ)-DLWE problem of dimension m states that the following holds: $(\mathbf{A}, \mathbf{sA} + \mathbf{e}) \overset{c}{\approx} (\mathbf{A}, \mathbf{u}) : \mathbf{A} \overset{\$}{\leftarrow} \mathbb{Z}_q^{n\times m}, \mathbf{s} \leftarrow \mathbb{Z}_q^n, \mathbf{e} \leftarrow \chi^m, \mathbf{u} \overset{\$}{\leftarrow} \mathbb{Z}_q^m$*

Extending the work of [ACPS09], where security is achieved for a secret $\mathbf{s} \leftarrow \chi^n$, the authors in [WZ17] show that for a noise distribution $\chi = \chi(\lambda)$, bounded by $\beta = \beta(\lambda)$, such that $\mathcal{H}_\infty(\chi) \geq \omega(\log \lambda)$, the following holds $(\mathbf{A}, \mathbf{SA} + \mathbf{E}) \overset{c}{\approx} (\mathbf{A}, \mathbf{U}) : \mathbf{A} \overset{\$}{\leftarrow} \mathbb{Z}_q^{n\times m}, \mathbf{S} \leftarrow \chi^{n\times n}, \mathbf{E} \leftarrow \chi^{n\times m}, \mathbf{U} \overset{\$}{\leftarrow} \mathbb{Z}_q^{n\times m}$ where $\|\mathbf{S}\|_\infty \leq \beta$, $\|\mathbf{E}\|_\infty \leq \beta$.

The GGH15 scheme [GGH15] encodes secrets along edges of a directed acyclic graph, where each node u associates a matrix $\mathbf{A}_u$ and a trapdoor t_u. Encoding a matrix $\mathbf{S}$ along $\mathbf{A}_u \rightsquigarrow \mathbf{A}_v$ is given by $\mathbf{C}_u = \mathbf{A}_u^{-1}(\mathbf{S}_u\mathbf{A}_v + \mathbf{E})$, where the notation $\mathbf{C}_u = \mathbf{A}_u^{-1}(\mathbf{Y})$ means $\mathbf{C}_u$ is a low-norm matrix such that $\mathbf{A}_u\mathbf{C}_u = \mathbf{Y}$. Multiplying encodings of $\mathbf{S}_u$, $\mathbf{S}_v$ along $\mathbf{A}_u \rightsquigarrow \mathbf{A}_v \rightsquigarrow \mathbf{A}_w$ satisfies the relation $\mathbf{A}_u\mathbf{C}_u\mathbf{C}_v = \mathbf{S}_u\mathbf{S}_v\mathbf{A}_w +$ (small error).

Wichs and Zirdelis [WZ17] restrict to a case where $\mathbf{S}$ is tensored with an identity matrix $\mathbf{I}_w \in \{0, 1\}^{w\times w}$ and GGH15 encoding of $\mathbf{I}_w \bigotimes \mathbf{S}$ is computed instead, where $\bigotimes$ denotes the Kronecker product of the matrices. They prove semantic security under the DLWE assumption: $(\mathbf{A}_u, (\mathbf{I}_w \bigotimes \mathbf{S})\mathbf{A}_v + \mathbf{E}) \overset{c}{\approx} (\mathbf{A}, \mathbf{U})$. The *directed encoding* scheme in [WZ17] encodes the same LWE secret $\mathbf{S} \in \mathcal{X}^{n\times n}$ along multiple paths $\{\mathbf{A}_0 \rightsquigarrow \mathbf{A}'_0, \ldots, \mathbf{A}_{w-1} \rightsquigarrow \mathbf{A}'_{w-1}\}$.

Lemma 3. *(Lattices with Trapdoors [Ajt99, GKPV10, MP12]) Let n, m and q satisfy the conditions in Definition 6. There exists a pair of PPT algorithms* $(\mathsf{TrapSamp}, \mathsf{SampPre})$ *defined as follows:*

- $(\mathbf{A}, t_\mathbf{A}) \leftarrow \mathsf{TrapSamp}(1^n, 1^m, q)$*: A randomized algorithm that samples a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ and trapdoor $t_\mathbf{A}$, where $q \geq 2$, $m > 2n \log q$*
- $\mathbf{C} \leftarrow \mathsf{SampPre}(\mathbf{A}, \mathbf{A}', t_\mathbf{A})$*: A pre-sampling algorithm that samples a low-norm matrix $\mathbf{C}$ such that $\mathbf{AC} = \mathbf{A}'$, where $\mathbf{A}, \mathbf{A}' \in \mathbb{Z}_q^{n\times m}$.*

Then for $q \geq 2$, $m > 2n \log q$, $n \geq 1$, the following distributions are statistically indistinguishable:

1. $\mathbf{A} \overset{s}{\approx} \widetilde{\mathbf{A}} \ : (\mathbf{A}, t_\mathbf{A}) \leftarrow \mathsf{TrapSamp}(1^n, 1^m, q), \widetilde{\mathbf{A}} \overset{\$}{\leftarrow} \mathbb{Z}_q^{n\times m}$.
2. $(\mathbf{A}, t_\mathbf{A}, \mathbf{C}) \overset{s}{\approx} (\mathbf{A}, t_\mathbf{A}, \widetilde{\mathbf{C}}) : \mathbf{C} \leftarrow \mathsf{SampPre}(\mathbf{A}, \mathbf{A}', t_\mathbf{A}), \mathbf{A}, \mathbf{A}' \leftarrow \mathbb{Z}_q^{n\times m}$*, "small" matrix* $\widetilde{\mathbf{C}} \leftarrow \mathbb{Z}_q^{m\times m}$.

At a high-level, the encoding scheme generates base matrix and trapdoor $(\mathbf{B}, t_\mathbf{B}) \leftarrow \mathsf{TrapSamp}(1^{w.n}, 1^m, q)$ and employs Algorithm 6 to calculate w encodings. The algorithms $\mathsf{TrapSamp}$, $\mathsf{SampPre}$ and Encode all access a random tape $\mathcal{T}$.

Algorithm 6 $\mathsf{Encode}(\mathbf{B}, \mathbf{B}', \mathbf{S}, t_{\mathbf{B}})$

```
1: Parse $\mathbf{B} = \begin{pmatrix} \mathbf{A}_0 \\ \vdots \\ \mathbf{A}_{w-1} \end{pmatrix}$ and $\mathbf{B}' = \begin{pmatrix} \mathbf{A}'_0 \\ \vdots \\ \mathbf{A}'_{w-1} \end{pmatrix}$, where $\mathbf{A}_i,\ \mathbf{A}'_i \in \mathbb{Z}_q^{n \times m}$
2: $\mathbf{E} \leftarrow \chi^{wn \times m}$, where $\mathbf{E} = \begin{pmatrix} \mathbf{E}_0 \\ \vdots \\ \mathbf{E}_{w-1} \end{pmatrix}$
3: Set $\mathbf{V} = (\mathbf{I}_w \bigotimes \mathbf{S})\, \mathbf{B}' + \mathbf{E}$
4: return $\mathbf{C} \leftarrow \mathsf{SampPre}(\mathbf{B}, \mathbf{V}, t_{\mathbf{B}})$
```

The compute-and-compare obfuscator encodes a function f which can be represented by a polynomial length *permutation branching program*, rather than any polynomial-sized circuit. Note that, any circuit of size $\mathrm{O}(L)$ and depth $\mathrm{O}(\log L)$ can be converted to permutation branching program of length polynomial in L.

Definition 7 (Permutation Branching Programs). *Let $L, w, \ell_{in} \in \mathbb{N}$, let $(x_1, \ldots, x_{\ell_{in}}) \in \{0,1\}^{\ell_{in}}$ be an input sequence. A permutation branching program of length L and width w computes a function $P : \{0,1\}^{\ell_{in}} \to \{0,1\}$, $(x_k)_{k \in \ell_{in}} \mapsto P((x_k)_{k \in \ell_{in}})$ based on a graph G defined as follows: G has $(L+1)w$ nodes grouped into $L+1$ levels of w nodes each, denoted by*

$$\{v_{i,j}\}_{i \in [L+1], j \in \{0,1,\ldots,w-1\}}.$$

At each level $i' \leq L$, one processes input variable $x_{I(i')}$ as follows: $v_{i',j}$ associates permutations $\pi_{i',0}(j)$, $\pi_{i',1}(j)$ which define the branch to walk to reach the nodes $v_{i'+1,j_1}$ and $v_{i'+1,j_2}$, $j_1 \neq j_2$. At input $(x_k)_{k \in \ell_{in}}$, the function starts from node $v_{1,0}$ (without loss of generality) at level 1, and at each level $i' \leq L$ follows permutation $\pi_{i',x_{I(i')}}(j_{i'})$ till it reaches the terminal node at level $L+1$ labeled by $v_{L+1,b}$, $b \in \{0,1\}$.

Consider a family of distribution ensembles $\mathcal{D} = \{\mathcal{D}_\lambda\}_{\lambda \in \mathbb{N}}$, where every $D \in \mathcal{D}_\lambda$ is poly-time samplable. Then $\mathcal{D}$ determines a program collection $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda \in \mathbb{N}} = \{f : \{0,1\}^{\ell_{in}(\lambda)} \to \{0,1\}^{\ell_{out}(\lambda)}\}$, where f is computable by ℓ_{out} polynomial-size permutation branching programs. The [WZ17] construction obfuscates $CC[f, \alpha]$ where $f \in \mathcal{F}$ and $\alpha \in \{0,1\}^{\ell_{out}}$, such that the obfuscation reveals no information about f, α. Hence $\mathcal{D}_\lambda$ must satisfy $\mathbf{H}_{\mathsf{HILL}}(\alpha|f) \geq nm \log q + \omega(\log \lambda)$.

The [WZ17] obfuscator works as follows: let $(P^{(k)})_{k \in [\ell_{out}]}$ be a sequence of permutation branching programs corresponding to each output bit of f, where the programs have a common length L and width w. For $1 \leq k \leq \ell_{out}$, sample matrices $\mathbf{A}_{i,j}^{(k)}$ with trapdoors $t_{i,j}^{(k)}$ corresponding to nodes $v_{i,j}^{(k)}$, where $i \leq L$, $0 \leq j < w$. At level $L+1$ of the branching program, select matrices such that $\mathbf{A}_{L+1,\alpha_1}^{(1)} + \cdots + \mathbf{A}_{L+1,\alpha_{\ell_{out}}}^{(\ell_{out})} = 0 \mod q$, where $\alpha = (\alpha_1, \ldots, \alpha_{\ell_{out}}) \in \{0,1\}^{\ell_{out}}$ is the target value. To achieve this, sample uniformly random matrices $\mathbf{A}_{L+1,j}^{(k)}$ and set $\mathbf{A}_{L+1,\alpha_{\ell_{out}}}^{(\ell_{out})} = -\sum_{k=1}^{\ell_{out}-1} \mathbf{A}_{L+1,\alpha_k}^{(k)}$. Next, sample secret low-norm matrices $\mathbf{S}_{i,0}$ and $\mathbf{S}_{i,1}$ with $\|\mathbf{S}_{i,b}\|_\infty \leq \beta$ which are the same across the ith levels of all the ℓ_{out} branching programs.

Encode the secret matrices into $\mathbf{C}_{i,0}^{(k)}$ and $\mathbf{C}_{i,1}^{(k)}$ following the *directed encoding scheme* discussed above, such that for an input $x = (x_1, \ldots, x_{\ell_{in}})$, the condition $\mathbf{A}_{1,0}^{(k)} \left(\prod_{i=1}^{L} \mathbf{C}_{i,x_{I(i)}}^{(k)}\right) \overset{c}{\approx} \left(\prod_{i=1}^{L} \mathbf{S}_{i,x_{I(i)}}\right) \mathbf{A}_{L+1,P^{(k)}(x)}^{(k)}$ is satisfied, where the common secret $\prod_{i=1}^{L} \mathbf{S}_{i,x_{I(i)}}$ is encoded along all the ℓ_{out} LWE samples. In the end, the obfuscator outputs $(\mathbf{A}_{1,0}^{(k)})$ and the encodings $(\mathbf{C}_{i,0}^{(k)}), \mathbf{C}_{i,1}^{(k)})$ for $i \in [L]$. The procedure is formally specified in Algorithm 7.

Algorithm 7 Obfuscator $\mathcal{O}_{CC}(1^\lambda, (P^{(k)})_{k\in[\ell_{out}]})$

```
1: for $k = 1$ to $\ell_{out}$ do
2:     Parse $P^{(k)} = (\pi^{(k)}_{i,b})_{i\in[L],b\in\{0,1\}}$; Sample $\mathbf{A}^{(k)}_{L+1,j} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$, $j \in \{0,\dots,w-1\}$
3:     if $k == \ell_{out}$ then
4:         $\mathbf{A}^{(\ell_{out})}_{L+1,\alpha_{\ell_{out}}} = -\sum_{l=1}^{\ell_{out}-1} \mathbf{A}^{(l)}_{L+1,\alpha_l}$
5:     end if
6:     Set $\mathbf{B}^{(k)}_{L+1} = \begin{pmatrix} \mathbf{A}^{(k)}_{L+1,0} \\ \vdots \\ \mathbf{A}^{(k)}_{L+1,w-1} \end{pmatrix}$
7:     for $i = 1$ to $L$ do
8:         Sample $(\mathbf{B}^{(k)}_i, t^{(k)}_i) \leftarrow \mathsf{TrapSamp}(1^{wn}, 1^m, q)$ with $\mathbf{B}^{(k)}_i = \begin{pmatrix} \mathbf{A}^{(k)}_{i,0} \\ \vdots \\ \mathbf{A}^{(k)}_{i,w-1} \end{pmatrix}$
9:     end for
10: end for
11: for $i = 1$ to $L$ do
12:     for $b = 0$ to $1$ do
13:         Sample $\mathbf{S}_{i,b} \leftarrow \mathcal{X}^{n\times n}$
14:         for $k = 1$ to $\ell_{out}$ do
15:             $\mathbf{C}^{(k)}_{i,b} \leftarrow \mathsf{Encode}\left(\mathbf{B}^{(k)}_i, \pi^{(k)}_{i,b}(\mathbf{B}^{(k)}_{i+1}), \mathbf{S}_{i,b}, t^{(k)}_i\right)$, where $\pi(\mathbf{B}^{(k)}_{i+1}) = \begin{pmatrix} \mathbf{A}^{(k)}_{i+1,\pi(0)} \\ \vdots \\ \mathbf{A}^{(k)}_{i+1,\pi(w-1)} \end{pmatrix}$
16:         end for
17:     end for
18: end for
19: return $\{\mathbf{A}^{(k)}_{1,0}\}_{k\in[\ell_{out}]}, \{(\mathbf{C}^{(k)}_{i,0}, \mathbf{C}^{(k)}_{i,1})\}_{k\in[\ell_{out}],i\in[L]}$
```

On input x, the evaluation procedure calculates $\mathbf{D}^{(k)} = \mathbf{A}^{(k)}_{1,0}\big(\prod_{i=1}^{n} \mathbf{C}^{(k)}_{i,x_{I(i)}}\big)$ and checks if $\sum_{k=1}^{\ell_{out}} \mathbf{D}^{(k)} \leq \beta\ell_{out}(2m\beta)^{L-1}$. For an accepting input ($f(x) = \alpha$), the summation of ℓ_{out} LWE samples reduces to $\sum_{k=1}^{\ell_{out}} \mathbf{E}^{(k)}$ as $\sum_{k=1}^{\ell_{out}} \mathbf{A}^{(k)}_{L+1,P^{(k)}(x)} = 0$. Since $\|\mathbf{E}^{(k)}\|_\infty \leq \beta\ell_{out}(2m\beta)^{L-1}$, the evaluation procedure allows the input. If $f(x) \neq \alpha$, the summation of the LWE samples will be uniformly random and contain large entries with high probability.

Wichs and Zirdelis prove that their compute-and-compare construction satisfies *correctness* ϕ_2.

4.2 Malicious Obfuscator for [WZ17]

We show a malicious obfuscator $\mathcal{A}_{CC}$ for the [WZ17] scheme that violates correctness ϕ_2. As already noted, unlike our prior constructions of malicious obfuscators, $\mathcal{A}_{CC}$ does not know the set of accepting inputs. Given $CC[f, \alpha]$ is evasive in nature, finding an accepting input has a probability negligible in λ. Furthermore, inverting f requires computations of order exponential in λ.

The malicious obfuscator takes an auxiliary input $y \in \{0,1\}^{\ell_{in}}$. Let $\alpha' = f(y)$. We think of y as a fixed backdoor that is used in all executions of the obfuscator (this corresponds to the aux in Definition 2). Since the function is evasive we have $\alpha' \neq \alpha$ with overwhelming probability. Hence y should not be accepted by the input program $CC[f, \alpha]$ with overwhelming probability. Recall that in the [WZ17] construction, the top-level matrices are sampled such that $\mathbf{A}^{(\ell_{out})}_{n+1,\alpha_{\ell_{out}}} = -\sum_{k=1}^{\ell_{out}-1} \mathbf{A}^{(k)}_{L+1,\alpha_k}$, while the remaining $\ell_{out}(w-1)$ matrices are uniformly random in $\mathbb{Z}_q^{n\times m}$. To ensure that y is accepted by the program, we impose an additional constraint $\sum_{k=1}^{\ell_{out}} \mathbf{A}^{(k)}_{L+1,\alpha'_k} = 0 \mod q$.

To do this, note that $\alpha' \neq \alpha$ implies there is some index $z \in [\ell_{out}]$ such that $\alpha_z \neq \alpha'_z$. The malicious obfuscator then chooses all matrices $\mathbf{A}^{(k)}_{L+1,j}$ for $k \neq z$ at random, and then solves $\mathbf{A}^{(z)}_{L+1,\alpha_z} = -\sum_{k\neq z} \mathbf{A}^{(k)}_{L+1,\alpha_k}$ and $\mathbf{A}^{(z)}_{L+1,\alpha'_z} = -\sum_{k\neq z} \mathbf{A}^{(k)}_{L+1,\alpha'_k}$. This ensures that $\mathcal{A}_{CC}$ accepts all correct inputs, along with y as $\sum_{k=1}^{\ell_{out}} \mathbf{D}^{(k)} \approx 0$ holds for either of the cases. We describe the steps formally in Algorithm 8.

Algorithm 8 Malicious Obfuscator $\mathcal{A}_{CC}\big(1^\lambda, (P^{(k)})_{k\in[\ell_{out}]}\big), y \in \{0,1\}^{\ell_{in}})$

```
1: if $(f(y) == \alpha)$ then
2:     return $\mathcal{O}_{CC}(1^\lambda, (P^{(k)})_{k\in[\ell_{out}]})$
3: else
4:     Set $\alpha' = f(y)$ ; Compute $z \xleftarrow{\$} \{1,\ldots,\ell_{out}\}$, s.t. $\alpha'_z \neq \alpha_z$
5:     for $k = 1$ to $\ell_{out}$ do
6:         Parse $P^{(k)} = (\pi^{(k)}_{i,b})_{i\in[L],b\in\{0,1\}}$ ; Sample $\mathbf{A}^{(k)}_{L+1,j} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$, $j \in \{0,\ldots,w-1\}$
7:         if $(k == z)$ then
8:             $\mathbf{A}^{(z)}_{L+1,\alpha_z} = -\sum_{l\neq z} \mathbf{A}^{(l)}_{L+1,\alpha_l}$ ; $\mathbf{A}^{(z)}_{L+1,\alpha'_z} = -\sum_{l\neq z} \mathbf{A}^{(l)}_{L+1,\alpha'_l}$
9:         end if
10:        Set $\mathbf{B}^{(k)}_{L+1}$ (line 6, Algorithm 7) ; Sample $(\mathbf{B}^{(k)}_i, t^{(k)}_i)$ (lines 7-9, Algorithm 7)
11:    end for
12:    Construct $\mathbf{C}^{(k)}_{i,j}$ as in lines 11-18 of Algorithm 7.
13:    return $\{\mathbf{A}^{(k)}_{1,0}\}_{k\in[\ell_{out}]}, \{(\mathbf{C}^{(k)}_{i,0}, \mathbf{C}^{(k)}_{i,1})\}_{k\in[\ell_{out}],i\in[L]}$
14: end if
```

Lemma 4. *Let $\lambda \in \mathbb{N}$ be the security parameter. Let $\mathcal{F} = \{\mathcal{F}_\lambda\}$ be an evasive compute-and-compare program collection parameterized by inputs of length $n = n(\lambda)$. Let $\alpha \in \{0,1\}^{\ell_{out}(\lambda)}$. Let $\mathcal{D} = \{\mathcal{D}_\lambda\}$, where $\mathcal{D}_\lambda$ is a distribution over evasive $\mathcal{F}_\lambda$. Let $(P^{(k)})_{k\in[\ell_{out}]}$ be a sequence of permutation branching programs that computes the evasive function $f \in \mathcal{F}$. Then algorithm $\mathcal{A}_{CC}$ (Algorithm 8) violates ϕ_2. Precisely, for every $y \in \{0,1\}^n$, there exists a $f \in \mathcal{D}_\lambda$ such that it is not true that for all $x \in \{0,1\}^n$ we have $\mathcal{A}_{CC}(1^\lambda, (P^{(k)})_{k\in[\ell_{out}]}, y)(x) = CC[f,\alpha](x)$.*

Proof. We show that $\mathcal{A}_{CC}$ violates ϕ_3, which implies violating correctness ϕ_2. Fix $y \in \{0,1\}^{\ell_{in}}$.

Since $\mathcal{D}_\lambda$ is a distribution on evasive compute-and-compare programs, there is some $f \leftarrow \mathcal{D}_\lambda$ such that $CC[f,\alpha](y)] = 0$. The malicious obfuscator ensures that $\mathcal{A}_{CC}(1^\lambda, (P^{(k)})_{k\in[\ell_{out}]}), y)(y) = 1$ with probability 1. The result follows for this choice of f and $x = y$. This shows that $\mathcal{A}_{CC}$ violates ϕ_3, which implies it violates ϕ_2. □

4.3 Indistinguishability of Obfuscators

We now prove that, under certain conditions, our malicious obfuscation cannot be detected by any distinguisher $\mathcal{B}$ who has knowledge of the program being obfuscated and who makes at most T adaptive queries to the obfuscator. In particular, our result requires ℓ_{out} to be large enough with respect to the LWE parameters (n, m, q) and the value of T. This is similar to the condition on ℓ_{out} that appears in Claim 4.12 of [WZ17]. Our proof follows a fairly similar approach as used in [WZ17] where we require $\mathcal{D}_\lambda$ to satisfy $\mathbf{H}_{\mathsf{HILL}}(\alpha|f) \geq nmT\log q + \omega(\log\lambda)$, such that the honest and malicious obfuscation distributions are statistically close, by the leftover-hash lemma.

Let $(P^{(k)})_{k\in[\ell_{out}]})$ be a chosen compute-and-compare program and let P' be a program output by the obfuscation oracle in the security game of Definition 2. As we have noted, just knowing $(P^{(k)})_{k\in[\ell_{out}]})$ does not imply knowledge of one or more inputs x such that $f(x) = \alpha$. However, let us assume that the distinguisher $\mathcal{B}$ does know some such inputs. Then the distinguisher can run the program P' on such inputs x and check they are accepted. The distinguisher can also choose some random x', check that $f(x') \neq \alpha$, and run $P'(x')$ to check that it rejects such inputs. Our malicious obfuscator has chosen an input y and defined $\alpha' = f(y)$. Hence the program will accept any input x' such that $f(x') = \alpha'$. Such malicious behaviour could be detected by $\mathcal{B}$ if there are many inputs that map under f to α'. This is why we require that f is evasive.

Theorem 2. *Let $\mathcal{D} = \{\mathcal{D}_\lambda\}_{\lambda\in\mathbb{N}}$ be a family of distribution ensembles, such that $(f,\alpha) \leftarrow \mathcal{D}_\lambda$ satisfies $\mathbf{H}_{\mathsf{HILL}}(\alpha|f) \geq nmT\log q + \omega(\log\lambda)$, where λ is the security parameter. Then for all PPT distinguishers $\mathcal{B}$, there exists a negligible function $\mu(\lambda)$, such that for all $\lambda \in \mathbb{N}$, for all pairs $\mathcal{A}_{CC}(1^\lambda,\cdot,y)$ and $\mathcal{O}_{CC}(1^\lambda,\cdot)$, the following holds:*

$$\left| \Pr_{y,\mathcal{B},\mathcal{A}_{CC}}\left[\mathcal{B}^{\mathcal{A}_{CC}(1^\lambda,\cdot,y)} = 1\right] - \Pr_{\mathcal{B},\mathcal{O}_{CC}}\left[\mathcal{B}^{\mathcal{O}_{CC}(1^\lambda,\cdot)} = 1\right] \right| \leq \mu(\lambda)$$

Proof. The only difference between the honest and malicious obfuscator is line 8 of Algorithm 8. Recall that the distinguisher can request up to T obfuscations of chosen circuits, including repeated obfuscations of the same circuit. Hence, for the worst case we assume that there are T requests to obfuscate the same f and α, and for which the malicious obfuscator would insert the same backdoor $\alpha' = f(y)$. The computation in line 8 is $\mathbf{A}^{(z)}_{L+1,\alpha'_z} = -\sum_{l\neq z}\mathbf{A}^{(l)}_{L+1,\alpha'_l} = \sum_{l\neq z}(\alpha'_l(\mathbf{A}^{(l)}_{L+1,0} - \mathbf{A}^{(l)}_{L+1,1}) - \mathbf{A}^{(l)}_{L+1,0}$. Following the same proof technique as [WZ17], we view $\mathbf{A}^{(z)}_{L+1,\alpha'_z}$ as the output of a universal hash function $h(\alpha'_1,\ldots,\alpha'_{\ell_{out}}) = \sum_{k=1}^{\ell_{out}-1}(\alpha'_k(\mathbf{A}^{(k)}_0 - \mathbf{A}^{(k)}_1) - \mathbf{A}^{(k)}_0)$ for certain matrices $\mathbf{A}^{(k)}_0, \mathbf{A}^{(k)}_1$ (we set $\mathbf{A}^{(z)}_b = 0$). As this is repeated up to T times, we concatenate all T output matrices. The output set of the hash function is thus $(\mathbb{Z}_q^{n\times m})^T$, with size q^{nmT}. This family is universal. Since, f,α is sampled from a distribution which satisfies $\mathbf{H}_{\mathsf{HILL}}(\alpha'_1,\ldots,\alpha'_{\ell_{out}}|f) \geq nmT\log q + \omega(\log\lambda)$, the statistical distance between the hash values (T maliciously formed matrices) and matrices chosen uniformly random in $(\mathbb{Z}_q^{n\times m})^T$ is at most $2^{-T\omega(\log\lambda)}$ (by the leftover-hash lemma), which is a negligible function in λ. □

4.4 Malicious Obfuscator for [GKVW20]

Goyal, Koppula, Vusirikala and Waters [GKVW20] show how to remove the evaluation errors in the [GKW17, WZ17] schemes, and obtain a perfectly correct obfuscation.

Algorithm 9 Obfuscator $\mathcal{O}_{PC}\big(1^\lambda,(P^{(k)})_{k\in[\ell_{out}]}\big)$

```
1: for $k = 1$ to $\ell_{out}$ do
2:     Parse $P^{(k)} = (\pi^{(k)}_{i,b})_{i\in[L],b\in\{0,1\}}$ ; $\mathbf{A}^{(k)}_{L+1,j} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$ ; $\mathbf{A}^{(k)}_{L+1,1-\alpha_k} \leftarrow \mathbf{A}^{(k)}_{L+1,1-\alpha_k} + \mathbf{D}$
3:     if $(k == \ell_{out})$ then
4:         $\mathbf{A}^{(\ell_{out})}_{L+1,\alpha_{\ell_{out}}} = -\sum_{l=1}^{\ell_{out}-1}\mathbf{A}^{(l)}_{L+1,\alpha_l}$
5:     end if
6:     Set $\mathbf{B}^{(k)}_{L+1}$ as line 6, Algorithm 7; Sample $(\mathbf{B}^{(k)}_i, t^{(k)}_i)$ as lines 7-9, Algorithm 7
7: end for
8: Construct $\mathbf{C}^{(k)}_{i,j}$ as in lines 11-18 of Algorithm 7.
9: return $\{\mathbf{A}^{(k)}_{1,0}\}_{k\in[\ell_{out}]}, \{(\mathbf{C}^{(k)}_{i,0}, \mathbf{C}^{(k)}_{i,1})\}_{k\in[\ell_{out}],i\in[L]}$
```

The authors claim that their construction, even though randomized, cannot be manipulated for inserting a secret backdoor. Nonetheless, we show how to cheat and design a malicious obfuscator for the [GKVW20] scheme. We present the construction in Algorithm 9, in the context of [WZ17].

Recall that for obfuscating $CC[f,\alpha]$, Wichs and Zirdelis sample the level $L+1$ matrices uniformly at random with the restriction $\sum_{k=1}^{\ell_{out}-1} \mathbf{A}^{(k)}_{L+1,\alpha_k} = 0 \mod q$. Then for an input x s.t. $f(x)=\alpha$, the LWE samples sum up to a matrix with small entries, and for $f(x)\neq\alpha$, at least of the $\mathbf{A}^{(k)}_{L+1,\alpha_k}$'s are uniformly random, and with a high probability outputs a matrix with large entries. To remove errors in evaluating the obfuscated program on non-accepting inputs, Goyal, Koppula, Vusirikala and Waters [GKVW20] impose an added constraint : sample ℓ_{out} matrices $\mathbf{A}^{(k)}_{L+1,1-\alpha_k}$ from a new distribution obtained by adding a fixed correction matrix $\mathbf{D}$ with "large" entries to the uniform distribution on $\mathbb{Z}_q^{n\times m}$. This guarantees that for any x with $f(x)\neq\alpha$, the summation of LWE samples always contains large entries and thus is rejected by the obfuscator.

Algorithm 10 Malicious Obfuscator $\mathcal{A}_{PC}(1^\lambda, (P^{(k)})_{k\in[\ell_{out}]}, y\in\{0,1\}^{\ell_{in}})$

```
1:  if $(f(y) == \alpha)$ then
2:      return $\mathcal{O}_{PC}(1^\lambda, (P^{(k)})_{k\in[\ell_{out}]})$
3:      Set $\alpha' = f(y)$ ; Compute $z \xleftarrow{\$} \{1,\dots,\ell_{out}\}$, s.t. $\alpha'_z \neq \alpha_z$
4:      for $k = 1$ to $\ell_{out}$ do
5:          Parse $P^{(k)} = (\pi^{(k)}_{i,b})_{i\in[L],b\in\{0,1\}}$ ; $\mathbf{A}^{(k)}_{L+1,j} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$; $\mathbf{A}^{(k)}_{L+1,1-\alpha_k} \leftarrow \mathbf{A}^{(k)}_{L+1,1-\alpha_k} + \mathbf{D}$
6:          if $(k == z)$ then
7:              $\mathbf{A}^{(z)}_{L+1,\alpha'_z} = -(\sum_{l\neq z}(\mathbf{A}^{(l)}_{L+1,\alpha'_l} + \mathbf{D}))$ ; $\mathbf{A}^{(z)}_{L+1,\alpha_z} = -\sum_{l\neq z}\mathbf{A}^{(l)}_{L+1,\alpha_l}$
8:          end if
9:          Set $\mathbf{B}^{(k)}_{L+1}$ as line 6, Algorithm 7; Sample $(\mathbf{B}^{(k)}_i, t^{(k)}_i)$ as lines 7-9, Algorithm 7
10:     end for
11:     Construct $\mathbf{C}^{(k)}_{i,j}$ as in lines 11-18 of Algorithm 7.
12:     return $\{\mathbf{A}^{(k)}_{1,0}\}_{k\in[\ell_{out}]}, \{(\mathbf{C}^{(k)}_{i,0}, \mathbf{C}^{(k)}_{i,1})\}_{k\in[\ell_{out}],i\in[L]}$
13: end if
```

We now design a malicious obfuscator $\mathcal{A}_{PC}$ that violates correctness ϕ_1 in the [GKVW20] scheme. In the default settings, the ℓ_{out} top-level matrices are sampled uniformly random, prior to adding $\mathbf{D}$. Note that, the hardness of LWE implies it is infeasible to distinguish $(\mathbf{A}, \mathbf{SA}+\mathbf{E})$ from $(\mathbf{A}, \mathbf{SA}+\mathbf{E}+\mathbf{D})$, and hence it is hard to determine whether the [GKVW20] obfuscator has been performed honestly.

We begin with the assumption that y is a bad input as otherwise we would have to return $\mathcal{O}_{PC}$. Since $\alpha'\neq\alpha$, there is at least one index $z\in[\ell_{out}]$ where they differ. The malicious obfuscator simply samples $\mathbf{A}^{(z)}_{L+1,\alpha'_z} = -(\sum_{l\neq z}(\mathbf{A}^{(l)}_{L+1,\alpha'_l} + \mathbf{D}))$, while the other matrices are sampled following Algorithm 9. The obfuscated program then allows good inputs, along with the secret backdoor y. We formalize the details in Algorithm 10.

5 Detectable Malicious Obfuscators

This section shows examples of randomized obfuscators that cannot be exploited to allow a master backdoor input. In particular, we discuss the obfuscation scheme for fuzzy matching under Hamming distance [GZ19]. We show that its correctness proof is incomplete, but a small tweak gives perfect correctness and so a backdoor cannot be inserted. We also briefly review the point function obfuscator [BMZ19], and show that the natural attempt to introduce a backdoor into their scheme is easily detected.

5.1 Reviewing the [GZ19] Construction

The hamming-ball membership obfuscator [GZ19] achieves DVBB security based on the *distributional decisional modular subset product* assumption.

Definition 8 (**Hamming-Ball Membership Programs**)**.** *Let $r, \ell \in \mathbb{N}$. Let $\mathsf{Ham}_{\alpha,r}$ denote a hamming-ball of radius $r < \ell$ around the center $\alpha \in \{0,1\}^\ell$. A hamming-ball membership program $P : \{0,1\}^\ell \to \{0,1\}$ on an input $x \in \{0,1\}^\ell$ is defined as*

$$P_{\mathsf{Ham}_{\alpha,r}}(x) = \begin{cases} 1\,, & \text{if } \Delta(\alpha, x) \le r \\ 0\,, & \text{otherwise.} \end{cases}$$

The authors restrict to "evasive" hamming-ball membership programs $\mathcal{P} = \{P_{\mathsf{Ham}_{\alpha,r}} : \alpha \leftarrow D\}$, where distribution $D \in \{0,1\}^\ell$ has high min-entropy.

Definition 9 (**Distributional Decisional Modular Subset Product Assumption** [GZ19])**.** *Let $\lambda \in \mathbb{N}$ be the security parameter and let r, ℓ be polynomials in λ, with $r < \frac{\ell}{2} - \sqrt{\ell\lambda \log 2}$. Let D be a distribution over $\{0,1\}^\ell$ with hamming-ball min-entropy λ. Let $B \in \mathbb{N}$ be such that $B = O(\ell \log(\ell))$. The distributional decisional modular subset product assumption states that the following distributions on $((p_i)_{i\in[\ell]}, q, A)$ are computationally indistinguishable: The first distribution samples $(\alpha_1, \dots, \alpha_\ell) \xleftarrow{\$} D$, $(p_1, \dots, p_\ell)$ a sequence of distinct primes sampled uniformly from $\{2, \dots, B\}$, q a uniformly sampled safe prime in $\{B^r, \dots, (1 + o(1))B^r\}$, and $A = \prod_{i=1}^{\ell} p_i^{\alpha_i} \mod q$. The second distribution samples $(p_1, \dots, p_\ell)$ and q in the same way, but samples A uniformly in $\mathbb{Z}_q^*$.*

For correctness the [GZ19] scheme uses an auxiliary point-function obfuscator $\mathcal{O}_{PT}$ (with ϕ_2 correctness) which encodes $c \in \{0,1\}^\ell$ in a point function $f : \{0,1\}^\ell \to \{0,1\}$ defined as $f_c(x) = 1$ if $x = c$, and 0 otherwise. The scheme works as follows: to encode α in $\mathsf{Ham}_{\alpha,r}$, sample distinct primes $(p_1, \dots, p_\ell)$ uniformly at random in $\{2, \dots, B\}$, together with a large prime modulus q such that $\prod_{i\in I} p_i < \frac{q}{2}$, for all $I \subset \{1, \dots, \ell\}$, $|I| \le r$. Finally, compute $A = \prod_{i=1}^{\ell} p_i^{\alpha_i} \mod q$ and $\mathcal{O}_{PT}(\alpha)$, and publish the values, along with the $\ell + 1$ primes. The formal procedure is given in Algorithm 11. On input $x \in \{0,1\}^\ell$, the evaluation procedure computes $X = \prod_{i=1}^{\ell} p_i^{x_i} \mod q$ and $E = AX^{-1} \mod q$. The idea is to recover the error vector $(e_i)_{i\in[\ell]} \in \{-1, 0, 1\}^\ell$ from $E = \prod_{i=1}^{\ell} p_i^{\alpha_i - x_i} \mod q$ using the convergents of the continued fraction representation of $\frac{E}{q}$. Note that, E can be expressed as $ND^{-1} \mod q$, where $N = \prod_{i=1}^{\ell} p_i^{u_i} \mod q$, $D = \prod_{i=1}^{\ell} p_i^{v_i} \mod q$, $u_i, v_i \in \{0,1\}$ and $u_i v_i = 0$ for all i. Then, there exists an $s \in \mathbb{Z}$, such that $ED = N + sq$ holds. By the theorem of Diophantine Approximation, $\frac{s}{D}$ is a convergent of $\frac{E}{q}$ when $ND < \frac{q}{2}$ (which happens when $P_{\mathsf{Ham}_{\alpha,r}}(x) = 1$). The algorithm $\mathsf{Factor}((p_i)_{i\in[\ell]}, k)$ returns a list of prime divisors of k as long as k is a product of primes in $(p_i)_{i\in[\ell]}$.

Algorithm 11 Obfuscator $\mathcal{O}_H(1^\lambda, P, \mathcal{O}_{PT})$

```
1: Sample distinct primes $(p_1, \dots, p_\ell)$ randomly from $\{2, \dots, B\}$, where $B = \mathrm{O}\big(\ell \log \ell\big)$
2: Sample safe prime $q$ such that $\prod_{i\in I} p_i < \frac{q}{2}$, for all $I \subset \{1, \dots, \ell\}$, $|I| \le r$
3: $A \leftarrow \prod_{i=1}^{\ell} p_i^{\alpha_i} \mod q$ ; $\alpha' \leftarrow \mathcal{O}_{PT}(\alpha)$
4: return $((p_1, \dots, p_\ell),\ q,\ A,\ \alpha')$
```

The [GZ19] scheme is not precise about their correctness guarantees. They show that every input within the Hamming ball $\mathsf{Ham}_{\alpha,r}$ is correctly accepted by the obfuscator, but they do not discuss whether every input outside the set is rejected with overwhelming probability. In fact, it seems that there will typically be some points just outside the boundary of the Hamming ball $\mathsf{Ham}_{\alpha,r}$ which will be accepted by the program (depending on the choice of $(p_1, \ldots, p_\ell)$). So the scheme does not have ϕ_2 correctness, but it probably has ϕ_3 correctness. Perfect correctness can be achieved by adding an extra check in line 10 of Algorithm 12: *check that the Hamming weight of e is $\leq r$.* This check prevents malicious obfuscation for the [GZ19] scheme: a distinguisher who knows α can check that $x = \alpha$ is accepted and, as long as the point function obfuscator is verifiable to be secure against malicious obfuscation, it follows that only values x of distance r from α are accepted.

Algorithm 12 Evaluation Eval(with embedded values $1^\lambda, (p_i)_{i\in[\ell]}, q, A, \alpha', \mathcal{O}_{PT}$)

```
1:  $X \leftarrow \prod_{i=1}^{\ell} p_i^{x_i} \mod q$ ; $E \leftarrow AX^{-1} \mod q$ ; $C \leftarrow \{\frac{h}{k} : \frac{h}{k}$ is a convergent of $\frac{E}{q}\}$
2:  for $\frac{h}{k} \in C$ do
3:    $e \leftarrow (0, \ldots, 0) \in \{0,1\}^\ell$ ; $F \leftarrow \mathsf{Factor}((p_i)_{i\in[\ell]}, k)$ ; $F' \leftarrow \mathsf{Factor}((p_i)_{i\in[\ell]}, kE \mod q)$
4:    if $F \neq \perp$ and $F' \neq \perp$ then
5:      for $i = 1$ to $\ell$ do
6:        if $p_i \in F \cup F'$ then
7:          $e_i \leftarrow 1$
8:        end if
9:      end for
10:     if $(\alpha' == \mathcal{O}_{PT}(x \oplus e))$ then
11:       return 1
12:     end if
13:   end if
14: end for
15: return 0
```

5.2 Reviewing the [BMZ19] Construction

The non-malleable point-function obfuscator by Bartusek, Ma and Zhandry [BMZ19] is ϕ_3 approx correct and DVBB secure in the GGM. Let $\mathbb{G} = \langle g \rangle$ be a group of prime order $q \in (2^{\lambda-1}, 2^\lambda)$. For a fixed $x \in \mathbb{Z}_q$, sample $a, b, c \xleftarrow{\$} \mathbb{Z}_q$ and publish $(a, b, c, g^{P(x)}, g^{Q(x)}, g^{R(x)})$ as the obfuscation of x, where $P(x) = ax + x^2 + x^3 + x^4 + x^5$, $Q(x) = bx + x^6$, $R(x) = cx + x^7$. The evaluation re-calculates the values and compares them with the published obfuscation. There are at most four roots $y \neq x$ of the fifth-degree polynomial $P(x)$, and for each such value, $(x^6 - y^6) + (x - y)b = 0$ and $(x^7 - y^7) + (x - y)c = 0$ with negligible probability by a union bound. Having a secret backdoor y would imply $P(x) = P(y)$, $Q(x) = Q(y)$ and $R(x) = R(y)$. The idea behind this being detectable is that a poly-time distinguisher who knows x, can compute all $y \neq x$ such that $P(x) = P(y)$. It can then check if $Q(x) = Q(y)$ and $R(x) = R(y)$. This should not happen (with overwhelming probability), otherwise the obfuscation is flagged as malicious. By the same argument, no malicious obfuscator exists for the Fenteany and Fuller's scheme [FF20].

6 Acknowledgments

We thank the Marsden Fund of the Royal Society of New Zealand for partly supporting this research.

References

[ACPS09] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In *Advances in Cryptology-CRYPTO 2009: 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, pages 595–618. Springer, 2009. doi:10.1007/978-3-642-03356-8_35.

[Ajt99] Miklós Ajtai. Generating hard instances of the short basis problem. In *Automata, Languages and Programming: 26th International Colloquium, ICALP'99 Prague, Czech Republic, July 11–15, 1999 Proceedings 26*, pages 1–9. Springer, 1999. doi:10.1007/3-540-48523-6_1.

[BBC+14] Boaz Barak, Nir Bitansky, Ran Canetti, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Obfuscation for evasive functions. In *Theory of Cryptography Conference, TCC 2014*, pages 26–51. Springer, 2014. doi:10.1007/978-3-642-54242-8_2.

[BGI+12] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im) possibility of obfuscating programs. *Journal of the ACM (JACM)*, 59(2):1–48, 2012. doi:10.1145/2160158.2160159.

[BGJS16] Saikrishna Badrinarayanan, Vipul Goyal, Aayush Jain, and Amit Sahai. Verifiable functional encryption. In *International Conference on the Theory and Application of Cryptology and Information Security, ASIACRYPT 2016*, pages 557–587. Springer, 2016. doi:10.1007/978-3-662-53890-6_19.

[BKM+18] Allison Bishop, Lucas Kowalczyk, Tal Malkin, Valerio Pastro, Mariana Raykova, and Kevin Shi. A simple obfuscation scheme for pattern-matching with wildcards. In *Advances in Cryptology–CRYPTO 2018: 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part III 38*, pages 731–752. Springer, 2018. doi:10.1007/978-3-319-96878-0_25.

[BLMZ19] James Bartusek, Tancrède Lepoint, Fermi Ma, and Mark Zhandry. New techniques for obfuscating conjunctions. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques, EUROCRYPT 2019*, pages 636–666. Springer, 2019. doi:10.1007/978-3-030-17659-4_22.

[BMZ19] James Bartusek, Fermi Ma, and Mark Zhandry. The distinction between fixed and random generators in group-based assumptions. In *Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part II 39*, pages 801–830. Springer, 2019. doi:10.1007/978-3-030-26951-7_27.

[BR17] Zvika Brakerski and Guy N Rothblum. Obfuscating conjunctions. *Journal of Cryptology*, 30(1):289–320, 2017. doi:10.1007/s00145-015-9221-5.

[CCK+22] Ran Canetti, Suvradip Chakraborty, Dakshita Khurana, Nishant Kumar, Oxana Poburinnaya, and Manoj Prabhakaran. COA-secure obfuscation and applications. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques, EUROCRYPT 2022*, pages 731–758. Springer, 2022. doi:10.1007/978-3-031-06944-4_25.

[Col18] Christian Collberg. Code obfuscation: Why is this still a thing? In *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, pages 173–174, 2018. doi:10.1145/3176258.3176342.

[Col23] C. Collberg. the tigress c obfuscator. https://tigress.wtf/index.html, 2023.

[CV09] Ran Canetti and Mayank Varia. Non-malleable obfuscation. In *Theory of Cryptography Conference, TCC 2009*, pages 73–90. Springer, 2009. doi:10.1007/978-3-642-00457-5_6.

[FF20] Peter Fenteany and Benjamin Fuller. Same point composable and nonmalleable obfuscated point functions. In *Applied Cryptography and Network Security: 18th International Conference, ACNS 2020, Rome, Italy, October 19–22, 2020, Proceedings, Part II 18*, pages 124–144. Springer, 2020. doi:10.1007/978-3-030-57878-7_7.

[For23] Irish Information Security Forum. pyobfgood - python obfuscation trap. https://www.iisf.ie/pyobfgood-blazestealer, 2023.

[GGH15] Craig Gentry, Sergey Gorbunov, and Shai Halevi. Graph-induced multilinear maps from lattices. In *Theory of Cryptography: 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II 12*, pages 498–527. Springer, 2015. doi:10.1007/978-3-662-46497-7_20.

[GKPV10] Shafi Goldwasser, Yael Tauman Kalai, Chris Peikert, and Vinod Vaikuntanathan. Robustness of the learning with errors assumption, 2010.

[GKVW20] Rishab Goyal, Venkata Koppula, Satyanarayana Vusirikala, and Brent Waters. On perfect correctness in (lockable) obfuscation. In *Theory of Cryptography Conference, TCC 2020*, pages 229–259. Springer, 2020. doi:10.1007/978-3-030-64375-1_9.

[GKW17] Rishab Goyal, Venkata Koppula, and Brent Waters. Lockable obfuscation. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 612–621. IEEE, 2017. doi:10.1109/focs.2017.62.

[GR07] Shafi Goldwasser and Guy N Rothblum. On best-possible obfuscation. In *Theory of Cryptography Conference, TCC 2007*, pages 194–213. Springer, 2007. doi:10.1007/s00145-013-9151-z.

[GZ19] Steven D Galbraith and Lukas Zobernig. Obfuscated fuzzy Hamming distance and conjunctions from subset product problems. In *Theory of Cryptography Conference, TCC 2019*, pages 81–110. Springer, 2019. doi:10.1007/978-3-030-36030-6_4.

[HMLS07] Dennis Hofheinz, John Malone-Lee, and Martijn Stam. Obfuscation for cryptographic purposes. In *Theory of Cryptography Conference, TCC 2007*, pages 214–232. Springer, 2007. doi:10.1007/s00145-009-9046-1.

[Int21] InterTrust. Whitecryption code protection. https://theiabm.org/wp-content/uploads/2021/02/whitecryption-code-protection-obfuscation-data-sheet.pdf, 2021.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques, EUROCRYPT 2012*, pages 700–718. Springer, 2012. doi:10.1007/978-3-642-29011-4_41.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):1–40, 2009. doi:10.1145/1568318.1568324.

[WZ17] Daniel Wichs and Giorgos Zirdelis. Obfuscating compute-and-compare programs under LWE. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 600–611. IEEE, 2017. doi:10.1109/focs.2017.61.