Auditable Obfuscation

Shalini Banerjee and Steven D. Galbraith

University of Auckland, New Zealand
{shalini.banerjee, s.galbraith}@auckland.ac.nz

Abstract. We introduce a new variant of *malicious obfuscation*. Our formalism is incomparable to the existing definitions by Canetti and Varia (TCC 2010), Canetti *et al.* (EUROCRYPT 2022) and Badrinar ayanan *et al.* (ASIACRYPT 2016). We show that this concept is natural and applicable to obfuscation-as-a-service platforms. We next define a new notion called *auditable obfuscation* which provides security against malicious obfuscation. Finally, we construct a proof of concept of the developed notions based on well-studied theoretical obfuscation proposals.

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 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.

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 obfuscation to third parties.

Canetti and Varia [CV09] introduced the notion of *verifiable obfuscation*, which is a related concept. They motivate their work by considering a login program that is intended to allow access to Alice, Bob, and Charles. They consider attacks to tamper with the correct execution of the program. Essentially their notion of verifiable obfuscation (which they also call verifiable non-malleability) allows an end user to check that an obfuscated program has not been tampered with by a third party. The obfuscator itself is still considered honest in their work.

Badrinarayanan, Goyal, Jain and Sahai [BGJS16] consider the case of a malicious obfuscator and aim to provide assurance to the user running the protocol, by allowing the user to check a predicate on the program. They give a solution for iO that triples the overhead (the user has to run three versions of the obfuscated program). Canetti, Chakraborty, Khurana, Kumar, Poburinnaya and Prabhakaran [CCK$^+$22] follow a similar notion where they use a perfectly binding commitment that attests some property of the unobfuscated circuit. The obfuscator returns a semi-functional circuit allowing the user to run a verification procedure and derive a fully-functional obfuscated circuit that satisfies the attested property using subexponentially-secure iO.

Note that the original developer knows the intended behaviour of the program, so one might think that they could easily verify the obfuscated program 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 than this. 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 obfuscation tool. Our main conceptual contribution is to explain that a formal definition of malicious obfuscation can be built using the existing notion of correctness (also called functionality preserving). Our second main contribution is to show a proof of concept for *auditable obfuscation*, which allows a software developer to audit the obfuscated code provided by a third party tool, and to check that the obfuscation process has not been malicious. 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. Along with that, we give examples of obfuscation schemes that cannot be leveraged for introducing 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.

1.1 Further Perspectives

Our main focus in this paper is to allow a software developer to check that a third-party obfuscator has behaved honestly. We do not focus on how a user can determine whether or not an obfuscated program is trustworthy (though this is also a challenging open question). But there is a third scenario to which our results can be applied, as we now explain.

Consider an app-store that distributes software and wants to maintain a reputation as a trusted supplier of good apps. However, for intellectual property reasons, the app-store may be selling obfuscated programs. How can the app-store know that these programs are not malicious for users?

The obfuscation service is provided by a third party and it may be the original app developer or the app-store who initiates the obfuscation. But the app-store itself wants to verify that the obfuscated program is trustworthy.

Our approach to auditable obfuscation allows the app-store to audit the obfuscation process, as long as the app-store is also provided with the original unobfuscated source code. This is because our audit concept requires verifying the steps taken by the obfuscator to convert the original source into the obfuscated code. This is where we deviate from Canetti and Varia [CV09], which do not require a verifier to know the un-obfuscated source. We also differ from [BGJS16, CCK+22] in that their verifier does not know the un-obfuscated code and is restricted to determining whether the obfuscated code satisfies some desirable public predicate. However, we do not think this is an issue in practice, since it is usually the end-users who are untrusted by the software developer, not the app-store.

A reader might be wondering whether it makes sense to consider a malicious obfuscator, since the obfuscator is always provided with the original program and so it can learn what it likes from the program and, for example, make its own copies to sell later. There are two responses to this critique. First, our methods apply even if the malicious obfuscator is a software tool that has been acquired by the software developer or app-store, and is always run in a protected sandbox and so is not able to transmit secrets back to headquarters. As we will show, such a tool could still be inserting a master backdoor without needing to interact with its owner or do any reverse-engineering of the program. Second, as we will explain in Section 5, reverse-engineering of the program may be difficult and it might not be possible for a malicious obfuscator to determine any behaviour of it (such as inputs that are accepted by the program). But we will show in the case of compute-and-compare obfuscation that a malicious obfuscator can still insert a master backdoor or other unwanted behaviour into the program.

1.2 Main Contributions

We now summarize our main contributions.

- The first contribution of this work is introducing the notion of *malicious obfuscators* that inject malicious functionality in the program in a way that is undetectable even by the original software developer. We argue that this concept is natural, since most developers use third party obfuscation providers.
- Our next contribution is in formulating *auditable obfuscation*, which provides security against malicious obfuscators. Our formalism builds on existing security definitions of obfuscation, while adding a verifiability property. As a proof of concept, we provide a general approach to auditable obfuscation, however this proof of concept is not very efficient as it essentially requires the verifier to re-do the obfuscation process. One can do slightly better by just checking some random subset of the steps, to get some assurance that

the obfuscation is honest. *We leave as an open problem for future work the development of more efficient methods.*

- Finally, we demonstrate 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], and Wichs and Zirdelis [WZ17]. We prove that our malicious obfuscators are indistinguishability from honest ones, even with respect to a verifier who knows the original un-obfuscated program. Additionally, 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. Following this, we formally define auditable obfuscation. 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 pseudocode.

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.* Hence, to prove that an obfuscated program has not been maliciously obfuscated it suffices to prove that the obfuscated program satisfies the same correctness as an output of the honest obfuscator.

Barak et al [BGI+12] first consider *perfect* correctness (also called functionality preserving), where the obfuscated program $\tilde{C}$ computes the exact same function as C. However, perfect correctness has turned out to be 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 list two of these formulations 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-size 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 $\tilde{C} = \mathcal{O}(C)$,*
  $$\forall x \in \{0,1\}^{n(\lambda)} : \ \tilde{C}(x) = C(x).$$
  - ϕ_2 : *For every $\lambda \in \mathbb{N}$ and every $C \in \mathcal{C}_\lambda$, there exists a negligible function $\mu(\lambda)$, such that:*
  $$\Pr_{\mathcal{O}} \left[\forall x \in \{0,1\}^{n(\lambda)} : \ \tilde{C}(x) = C(x) \right] > 1 - \mu(\lambda)$$
  where the probability is over the coin tosses of $\mathcal{O}$ in computing $\tilde{C} = \mathcal{O}(C)$. This is formulated in Definition 4.1 of [BGI+ 12].
  - ϕ_3 : *For every $\lambda \in \mathbb{N}$ and for every $x \in \{0,1\}^{n(\lambda)}$ and every C, there exists a negligible function $\mu(\lambda)$, such that:*
  $$\Pr_{\mathcal{O}} \left[\tilde{C}(x) = C(x) \right] > 1 - \mu(\lambda)$$
  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 $q(|C|)$, 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 $D \in \mathcal{D}_\lambda$:*
$$\left| \Pr_{C \leftarrow \mathcal{D}_\lambda, \mathcal{O}, \mathcal{A}} [\mathcal{A}(\mathcal{O}(C)) = 1] - \Pr_{C \leftarrow \mathcal{D}_\lambda, \mathcal{S}} [\mathcal{S}^C(1^\lambda) = 1] \right| \leq \mu(\lambda)$$
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 outstanding 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 $\tilde{C}(y) = 1$).

In addition, we require that a polynomial distinguisher cannot detect the fact 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.

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 $C \in \mathcal{C}_\lambda$)*
$$\left| \Pr_{\mathsf{aux},\mathcal{B},\mathcal{A}} [\mathcal{B}^{\mathcal{A}(1^\lambda,\cdot,\mathsf{aux})} = 1] - \Pr_{\mathcal{B},\mathcal{O}} [\mathcal{B}^{\mathcal{O}(1^\lambda,\cdot)} = 1] \right| \le \mu(\lambda)$$
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 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.

2.2 Formalizing Auditability in Obfuscators

We now define the notion of auditable obfuscation. Recall from the earlier discussions, our goal behind introducing auditable obfuscation is in defending against malicious obfuscators. Furthermore, we restrict to an efficient verifier (auditor) who knows the program being obfuscated. Crucially, we require an auditable obfuscation scheme to inherit the properties given in Definition 1, while incorporating an auditability property that ensures the correctness of the obfuscation.

There is some relevant previous work in this area. Zobernig, Galbraith and Russello [ZGR19] present a construction for verifying whether a seemingly opaque predicate is triggered by a secret input known to the obfuscator such that some potentially malicious code gets activated. The verification procedure employs cryptographic hash functions. Their work shows a simple case of verifiability.

At a high-level, *auditable obfuscation* ($\mathcal{AO}$) is a two-step process: The first step is carried out by an obfuscator, who runs an efficient algorithm $\mathcal{AO}.\mathsf{Obf}$, that takes as input a program C, and outputs an obfuscated program $\tilde{C}$ along with a proof π. Informally, π will allow an auditor to verify the correctness of obfuscation. The second step is performed by a verifier (auditor), who knows C. Precisely, $\mathcal{AO}.\mathsf{Verify}$ is an efficient algorithm, such that $\mathcal{AO}.\mathsf{Verify}\,(1^\lambda, C, \tilde{C}, \pi) = 1$, if $\tilde{C}$ is the correct obfuscation of C. We consider an honest obfuscator $\mathcal{O}$ with correctness ϕ_i and we require: (i) if $\tilde{C}$ satisfies ϕ_i, then $\mathcal{AO}.\mathsf{Verify}\,(1^\lambda, C, \tilde{C}, \pi) = 1$, and (ii) given a purported obfuscation $(\tilde{C}, \pi) \leftarrow \mathcal{A}(1^\lambda, C)$, if $\mathcal{AO}.\mathsf{Verify}\ (1^\lambda, C, \tilde{C}, \pi) = 1$, then $\tilde{C}$ satisfies ϕ_i.

We now give the formal definition of auditable obfuscation.

Definition 3 (Auditable Obfuscation). *Let $\lambda \in \mathbb{N}$ be the security parameter of the system. For any family of polynomial-size programs $\mathcal{C} = \{\mathcal{C}_\lambda\}$ and distribution ensembles $\mathcal{D} = \{\mathcal{D}_\lambda\}$, let $\mathcal{O}$ be an honest obfuscator that satisfies the conditions given in Definition 1 with correctness $\phi_i \in \phi$. Then auditable obfuscation $\mathcal{AO}$ for the family $\mathcal{C}$ and distribution class $\mathcal{D}$ is a pair of PPT algorithms* $(\mathcal{AO}.\mathsf{Obf}, \mathcal{AO}.\mathsf{Verify})$ *such that for every $\lambda \in \mathbb{N}$ and every $C \in \mathcal{C}_\lambda$:*

- *For every $(\tilde{C}, \pi) \leftarrow \mathcal{AO}.\mathsf{Obf}\ (1^\lambda, C)$, $\ \mathcal{AO}.\mathsf{Verify}\ (1^\lambda, C, \tilde{C}, \pi) \rightarrow \{0, 1\}$.*

- *(Soundness) : For every PPT adversary $\mathcal{A}$, if $[(\tilde{C}, \pi) \leftarrow \mathcal{A}\ (1^\lambda, C)] \wedge [1 \leftarrow \mathcal{AO}.\mathsf{Verify}\ (1^\lambda, C, \tilde{C}, \pi)]$, then $\tilde{C}$ satisfies ϕ_i.*

3 Generic Solution for Auditable Obfuscation

The obfuscators we will study are randomised and make a large number of random choices during the obfuscation. The attacks we will show on these schemes involve replacing some of the random values with specially chosen values to introduce malicious functionality.

Hence, there is a simple solution to auditable obfuscation: First, de-randomise the obfuscation, by arranging that all random choices are generated using a pseudorandom number generator on an initial randomly chosen seed. Then the proof of correct obfuscation is simply the value of the seed. The verifier (auditor) simply re-computes the obfuscation using the same pseudorandom generator initialised with that seed value, and checks if the output is the same.

Formally, the auditable obfuscator $\mathcal{AO}(1^\lambda, C)$ generates a random seed seed and generates an arbitrarily long sequence of pseudorandom bits using a pseudorandom generator $\mathsf{PRG}(\mathsf{seed})$. Then $\mathcal{AO}$ calls the original (honest) obfuscation algorithm $\mathcal{O}$, where all random numbers are generated using the output of the PRG. The output of $\mathcal{AO}$ is the obfuscated program $\tilde{C} = \mathcal{O}(C)$ generated by $\mathcal{O}$, together with the proof $\pi = \mathsf{seed}$. The auditor $\mathcal{AO}.\mathsf{Verify}(1^\lambda, C, \tilde{C}, \mathsf{seed})$ simply uses the same PRG on input seed and runs the obfuscator $\mathcal{O}$ to get a program C'. The auditor returns 1 if and only if C' is equal to the value $\tilde{C}$ given as input to the auditor.

This approach will satisfy the requirements for auditable obfuscation. Essentially, all a malicious obfuscator can do is bias the choice of randomness (e.g., by running the obfuscation polynomially-many times for different choices of seed until getting a program $\tilde{C} = \mathcal{O}(C)$ with the desired properties), but this is not enough to violate the correctness guarantee from the original obfuscator. We prove the soundness of this approach in the following meta-theorem.

Theorem 1. *Let $\mathcal{O}$ be an obfuscator with correctness ϕ and let $\mathcal{AO}$ be the generic auditable obfuscator constructed above. Let PRG be a pseudorandom generator that is indistinguishable from uniform. Then for every PPT adversary $\mathcal{A}$ and every $(\tilde{C}, \mathsf{seed}) \leftarrow \mathcal{A}(1^\lambda, C)$, if $\mathcal{AO}.\mathsf{Verify}(1^\lambda, C, \tilde{C}, \mathsf{seed}) = 1$, then $\tilde{C}$ satisfies correctness ϕ.*

Proof. Towards a contradiction, we suppose the obfuscated program $\tilde{C}$ does not satisfy ϕ. The below case handles the case ϕ_3, as the other cases are similar. This means that for fixed C and x

$$\underset{\mathcal{A}}{Pr}\ [\, \mathcal{A}(1^\lambda, C)(x) \neq C(x)\,] > \epsilon(\lambda)$$

where $\epsilon(\lambda)$ is a noticeable function in λ.

Since the purported obfuscation $\tilde{C} = \mathcal{A}(1^\lambda, C)$ is accepted by $\mathcal{AO}.\mathsf{Verify}$, it follows that the honest obfuscator $\mathcal{O}$ could have generated the same program $\tilde{C}$ by choosing the same seed seed. But, $\mathcal{A}$ could have tried at most polynomially many seeds, say $p(\lambda)$, to bias the output. This means the honest obfuscator could also have generated the same output with probability $\frac{1}{p(\lambda)}$. This implies that running $\mathcal{O}$ once yields

$$\underset{\mathcal{O},C}{Pr}\ [\, \mathcal{O}(1^\lambda, C)(x) \neq C(x)\,] > \frac{\epsilon(\lambda)}{p(\lambda)}$$

which is still a noticeable function in λ. This contradicts the assumption that $\mathcal{O}$ has correctness ϕ. □

We should emphasise that this security result only ensures that the obfuscated program has the same approximate correctness guarantee as the honest scheme. This does not mean that the obfuscated program is perfectly correct, or that the bad inputs are not biased in some way. It just means that a malicious obfuscator cannot be much different than a random output of the original obfuscator.

Our solution is not very efficient. Ideally the proof would be short and such that the verification algorithm is faster than running the obfuscation. Our above scheme at least has a short proof. It is an open problem to obtain an efficient solution. However we do remark that the verifier could simply check a random subset of the operations performed by the obfuscator, and accept the program if they are all computed correctly. This might be sufficient to detect malicious behaviour in some situations. *One other comment on this problem*: Our approach requires the auditor to understand the obfuscation tool. In practice, the obfuscation tool might have its own IP that it wishes to protect, and so it may not wish to make its processes transparent. Clearly there are many challenges for future work in this area.

4 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.

4.1 Reviewing the [BKM+18] Construction

We 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 4 (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*

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

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 n. If the ith bit of the pattern is j, where $j \in \{0,1\}$ or if $\mathsf{pat}_i = \star$, then evaluate the 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

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

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

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 field element in the exponent is returned by the algorithm, and the evaluator correctly rejects the input with overwhelming probability. The procedure is formally described in Algorithm 2.

Algorithm 2 Evaluation (with $2n$ embedded values $(h_{i,j})_{i \in [n], j \in \{0,1\}}$)

```
Input: $n \in \mathbb{N}$, $x \in \{0,1\}^n$
Output: 0 or 1.
1: for $i = 1$ to $n$ do
2:     $\gamma_i \leftarrow \prod_{j \neq i} \frac{2j - x_j}{2i - x_i - x_j + 2j}$
3: end for
4: Compute $T = \prod_{i=1}^{n} (h_{i,x_i})^{\gamma_i}$
5: if $(T == g^0)$ then
6:     return 1
7: else
8:     return 0
9: end if
```

4.2 Malicious Obfuscator for [BKM+18]

As is clear from Definition 4, 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 $\mathrm{O}\big(n\big)$ 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 2 positions (this is the case with overwhelming probability for evasive conjunctions). So we start with a string $y \in \{0, 1\}^n$ such that $C(y) = 0$ and y does not match the pattern in at least *two* positions. 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. We know 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

Algorithm 3 Malicious Obfuscator $\mathcal{A}_{Con}(1^\lambda, C, y \in \{0,1\}^n$ s.t. $C(y) = 0)$

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

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 accepts all inputs satisfying the pattern, as well as the input y with probability 1. 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 3.

Lemma 1. *The program output by Algorithm 3 accepts y and all inputs x that satisfy the pattern. Algorithm 3 violates ϕ_3.*

Proof. On input x satisfying the pattern, the correctness follows from the correctness result by Bishop *et al.* [BKM$^+$18], since the values h_{i,x_i} used in Algorithm 2 are the same.

On input y, when $i \in \mathcal{L}_y$ the values h_{i,y_i} are equal to $g^{H(2i+y_i)}$, while when $i \notin \mathcal{L}_y$ the values h_{i,y_i} are equal to $g^{F(2i+y_i)}$. But $H(2i + y_i) = F(2i + y_i)$ on those values, the Lagrange interpolation again computes $g^{H(0)} = g^0$ and so y is accepted. Since y is always accepted, but satisfies $C(y) = 0$ for most C, this shows that the malicious obfuscator does not satisfy correctness ϕ_3. $\square$

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

4.3 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 $2n$. We start with a high-level overview of the scheme, followed by their formal descriptions (Algorithms 4 and 5).

Definition 5. *Let $\mathbf{B}$ be the $(n+1)\times 2n$ dimensional matrix*

$$\begin{pmatrix} 1 & 2 & \dots & 2n \\ 1 & 2^2 & \dots & (2n)^2 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & 2^{n+1} & \dots & (2n)^{n+1} \end{pmatrix}.$$

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

Algorithm 4 Obfuscator $\mathcal{O}_{Dual}(1^\lambda, C)$

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

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 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.

Algorithm 5 Evaluation Eval (with embedded data $1^\lambda, \mathbf{B} \in \mathbb{Z}_q^{(n+1)\times 2n}$, $\mathbf{v}$)

Input: $x \in \{0, 1\}^n$
Output: 0 or 1

```
1: Define $\mathbf{B}_x \in \mathbb{Z}_q^{(n+1)\times n}$, where column $j$ is set to $(\mathbf{B}_x)_j = \mathbf{B}_{2j-x_j}$
2: Solve for non-zero vector $\mathbf{t} \in \mathbb{Z}_q^{1\times(n+1)}$ such that $\mathbf{tB}_x = 0$
3: if ($\prod_{i=1}^{n+1} \mathbf{v}_i^{\mathbf{t}_i} == g^0$) then
4:     return 1
5: else
6:     return 0
7: end if
```

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, which informally states that for any given input, the obfuscation is correct with overwhelming probability over the coin tosses of the obfuscator and the program (see Definition 1).

4.4 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 4) 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 $\mathbf{tB}$ that also has the correctly structured zero entries.

Let $\mathbf{E} \in \mathbb{Z}_q^{2n\times 1}$ be the subspace of vectors with basis $\{\mathbf{e}_{2i-(1-b)} : \mathsf{pat}_i = b\}$. Let $\mathbf{E}' = \{\mathbf{w} : \mathbf{w} \in \mathbf{E} \wedge \mathbf{t}_y\mathbf{B}\mathbf{w} = 0\}$. Since $n+w$ are fixed zero entries, the dimension of $\mathbf{E}'$ is $n-w-1$. Thus, for an input y that does not match pat, if the obfuscator selects a vector $\mathbf{e}^*$ from $\mathbf{E}'$ and publishes $\mathbf{Be}^*$, the evaluation algorithm will accept y, as $\mathbf{t}_y\mathbf{Be}^* = 0$. We specify the procedure formally in Algorithm 6.

Algorithm 6 Malicious Obfuscator $\mathcal{A}_{Dual}(1^\lambda, C, y \in \{0,1\}^n)$

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

Lemma 2. *The program output by Algorithm 6 accepts y and all inputs x that satisfy the pattern. Algorithm 6 violates ϕ_3.*

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 $x \in \{0,1\}$

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

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

Now consider $\mathcal{A}_{Dual}$ with some fixed master backdoor y. Let C be such that $C(y) = 0$ (which is overwhelmingly the case if C corresponds to a uniformly sampled pattern). Then $\mathcal{O}_{Dual}(1^\lambda, C)(y) = 1$, and so

$$\Pr_{\mathcal{O}_{Dual}} [\, \mathcal{O}_{Dual}(1^\lambda, C)(y) = C(y) \,] = 0 \not> 1 - \mu(\lambda)$$

This proves that $\mathcal{A}_{Dual}$ violates ϕ_3. □

4.5 Indistinguishability of our Malicious Obfuscator

We now prove computational indistinguishability of the malicious and honest obfuscators (specified in Algorithms 4 and 6) in the generic group model. The generic group model 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. 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 2. *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 7. 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.

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 $q > 2^{2\lambda}$.

Algorithm 7 Oracle handler

```
$\mathcal{O}(C)$ query:
 1: Initialize $\mathbf{e}^{(t)} \in R^{2n}$, where $R = \mathbb{Z}_q[X_1^{(t)}, \ldots, X_n^{(t)}]$, to be the zero vector
 2: for $i = 1$ to $n$ do
 3:     if ($\mathsf{pat}_i == b$) then
 4:         $\mathbf{e}^{(t)}_{2i-(1-b)} \leftarrow X_i^{(t)}$
 5:     end if
 6: end for
 7: Compute $(F_1, \ldots, F_{n+1}) = \mathbf{B}\mathbf{e}^{(t)} \in R^{n+1}$
 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: Let $F_i, F_j \in \mathbb{Z}_q[X_1^{(1)}, \ldots, X_n^{(T)}]$ be such that $(\sigma_i, F_i) \in \mathcal{L}$ and $(\sigma_j, F_j) \in \mathcal{L}$ (if they
    don't exist return $\perp$)
 2: Let $F = F_i - F_j$
 3: if $\exists\sigma : (\sigma, F) \in \mathcal{L}$ then
 4:     Return $\sigma$
 5: else
 6:     $\sigma \xleftarrow{\$} \{0,1\}^\tau$
 7:     $\mathcal{L} \leftarrow \mathcal{L} \cup \{(\sigma, F)\}$
 8:     Return $\sigma$
 9: end if

$\boxed{\mathsf{isZero}(\sigma)\ \textbf{query}}$
 1: Let $F \in \mathbb{Z}_q[X_1^{(1)}, \ldots, X_n^{(T)}]$ be such that $(\sigma, F) \in \mathcal{L}$ (return $\perp$ if none exists)
 2: if $(F == 0)$ then
 3:     Return True
 4: else
 5:     Return False
 6: end if
```

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 $(x_1^{(t)}, \dots, x_n^{(t)}) \in \mathbb{Z}_q^n$, 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 $F_0(X_1^{(t)}, \dots, X_n^{(t)})$ that correspond to the condition $\mathbf{t}_y \mathbf{B} \mathbf{e}^{(t)} = 0$. The simulation chooses the point $(x_1^{(t)}, \dots, x_n^{(t)}) \in \mathbb{Z}_q^n$ uniformly at random. Again, the only things we need to be concerned about are whether $F_0(x_1^{(t)}, \dots, x_n^{(t)}) = 0$ 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

$$p = \Pr_{\mathcal{B}, \mathcal{O}_{Dual}} [\, \mathcal{B}^{\mathcal{O}_{Dual}(1^\lambda, \cdot)} = 1]$$

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

$$\Pr_{\mathcal{B}, \mathcal{O}_{Dual}} [\, \mathcal{B}^{\mathcal{O}_{Dual}(1^\lambda, \cdot)} = 1] = p\, Pr[\neg \mathsf{Fail}] + Pr[\mathcal{B} = 1 | \mathsf{Fail}]\, Pr[\mathsf{Fail}] = p + \mu_1(\lambda)$$

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

$$Pr\,[\, \mathcal{B}^{\mathcal{A}_{Dual}} = 1 | \neg \mathsf{Fail}' \,] = p.$$

It follows that

$$Pr\,[\, \mathcal{B}^{\mathcal{A}_{Dual}} = 1 \,] = p + \mu_2(\lambda)$$

for some negligible function $\mu_2(\lambda)$. This completes the proof. □

4.6 Auditable Obfuscator for [BLMZ19]

We now design the auditable obfuscator $\mathcal{AO}_{Dual}(1^\lambda, C)$ that provides security against the malicious obfuscator $\mathcal{A}_{Dual}(1^\lambda, C, y)$. We follow the construction of Section 3, namely to de-randomise the obfuscation and to provide the seed for the PRG as the proof π that the obfuscation has been correctly followed.

To be precise, $\mathcal{O}_{Dual}(1^\lambda, C)$ is expected to choose the non-zero entries randomly from $\mathbb{Z}_q$, so our auditable obfuscator $\mathcal{AO}_{Dual}$ uses the PRG output to generate all such group elements. The algorithm $\mathcal{AO}_{Dual}.\mathsf{Verify}$ simply recomputes the obfuscation using the given seed. By Theorem 1, if the verify algorithm accepts then the obfuscated program has the intended correctness and no malicious behaviour has been introduced.

5 Malicious Obfuscation 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. Our techniques apply to both schemes, but we only present the details for the obfuscation scheme by Wichs and Zirdelis [WZ17], as this gives the main idea for both 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 program C in the clear.

We first recall the definition of compute-and-compare programs.

Definition 6 (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.

5.1 Reviewing the [WZ17] Construction

We now review the compute-and-compare construction by Wichs and Zirdelis [WZ17] which achieves DVBB security under the *learning with errors* (LWE) assumption.

For obfuscating this class of programs, the [WZ17] construction employs GGH15 encodings by Gentry, Gorbunov and Halevi [GGH15] in a restricted setting. We state the LWE problem by Regev [Reg09].

Definition 7 (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 distributions are computationally indistinguishable:*

$$(\mathbf{A}, \mathbf{sA} + \mathbf{e}) \overset{c}{\approx} (\mathbf{A}, \mathbf{u}) : \mathbf{A} \overset{\$}{\leftarrow} \mathbb{Z}_q^{n \times m}, \mathbf{s} \leftarrow \mathbb{Z}_q^n, \mathbf{e} \leftarrow \chi^m, \mathbf{u} \overset{\$}{\leftarrow} \mathbb{Z}_q^m$$

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 Decisional LWE 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, q satisfy the conditions in Definition 7. There exists a pair of PPT algorithms* (TrapSamp, 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. $\mathbf{A} \overset{s}{\approx} \widetilde{\mathbf{A}}$: $(\mathbf{A}, t_{\mathbf{A}}) \leftarrow \mathsf{TrapSamp}(1^n, 1^m, q)$, $\widetilde{\mathbf{A}} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$.
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* $\tilde{\mathbf{C}} \leftarrow \mathbb{Z}_q^{m\times m}$.

At a high-level, the encoding scheme generates base matrix and trapdoor $(\mathbf{B}, t_{\mathbf{B}}) \leftarrow \mathsf{TrapSamp}(1^{w \cdot n}, 1^m, q)$ and employs Algorithm 8 to calculate w encodings.

Algorithm 8 $\mathsf{Encode}(\mathbf{B}, \mathbf{B}', \mathbf{S}, t_{\mathbf{B}})$

```
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 algorithms TrapSamp, SampPre and Encode all access a random tape $\mathcal{T}$. Later, we will de-randomize these algorithms by replacing $\mathcal{T}$ with a pseudorandom generator PRG.

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. In what follows, we define permutation branching programs.

Definition 8 (Permutation branching Programs). *Let $L, w, \ell_{in} \in \mathbb{N}$ and 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 $P : \{0,1\}^{\ell_{in}} \to \{0,1\}$, $(x_1, \ldots, x_{\ell_{in}}) \mapsto P(x_1, \ldots, x_{\ell_{in}})$ 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*

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

For $1 \leq i \leq L+1$ define $I(i) \in \{1, \ldots, \ell_{in}\}$ to be such that $I(i) \equiv i \mod \ell_{in}$. 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 $(x_1, \ldots, x_{\ell_{in}})$, 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 polynomial-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 $\mathbf{H}_{\mathsf{HILL}}(\alpha|f) \geq \gamma(\lambda, L)$, where γ exceeds some polynomial threshold in λ and length L of the permutation branching program. Concretely, $\gamma > nm\log q + \omega(\log\lambda)$ ensures that $CC[f,\alpha]$ is *evasive*.

Precisely, the compute-and-compare 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 $\mathbf{A}^{(k)}_{i,j}$ with trapdoors $t^{(k)}_{i,j}$ corresponding to nodes $v^{(k)}_{i,j}$, where $i \leq L$, $0 \leq j < w$. At level $L+1$ of the branching program, select matrices such that $\mathbf{A}^{(1)}_{L+1,\alpha_1} + \cdots + \mathbf{A}^{(\ell_{out})}_{L+1,\alpha_{\ell_{out}}} = 0 \pmod q$, where $\alpha = (\alpha_1,\ldots,\alpha_{\ell_{out}}) \in \{0,1\}^{\ell_{out}}$ is the target value. To achieve this, sample uniformly random matrices $\mathbf{A}^{(k)}_{L+1,j}$ and set $\mathbf{A}^{(\ell_{out})}_{L+1,\alpha_{\ell_{out}}} = -\sum_{k=1}^{\ell_{out}-1}\mathbf{A}^{(k)}_{L+1,\alpha_k}$. 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 $\mathbf{C}^{(k)}_{i,0}$ and $\mathbf{C}^{(k)}_{i,1}$ following the *directed encoding scheme* discussed above, such that for an input $x = (x_1,\ldots,x_{\ell_{in}})$, the following condition is satisfied:

$$\mathbf{A}^{(k)}_{1,0}\left(\prod_{i=1}^{L}\mathbf{C}^{(k)}_{i,x_{I(i)}}\right) \overset{c}{\approx} \left(\prod_{i=1}^{L}\mathbf{S}_{i,x_{I(i)}}\right)\mathbf{A}^{(k)}_{L+1,P^{(k)}(x)}$$

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 $(\mathbf{A}^{(k)}_{1,0})$ and the encodings $(\mathbf{C}^{(k)}_{i,0}), \mathbf{C}^{(k)}_{i,1})$ for $i \in [L]$. The procedure is formally specified in Algorithm 9.

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 whether $\sum_{k=1}^{\ell_{out}}\mathbf{D}^{(k)} \leq \beta\ell_{out}(2m\beta)^{L-1}$. Note that, 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. We give the details in Algorithm 10.

Wichs and Zirdelis prove that their compute-and-compare construction satisfies *correctness* ϕ_2, which states that for every $(f,\alpha) \leftarrow \mathcal{D}_\lambda$ the obfuscation is correct on all inputs $x \in \{0,1\}^{\ell_{in}}$ except with negligible probability over the coin tosses of the obfuscator.

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

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

Algorithm 10 Evaluation Eval (with embedded values $1^\lambda, (\mathbf{A}^{(k)}_{1,0}), (\mathbf{C}^{(k)}_{i,0}, \mathbf{C}^{(k)}_{i,1})$) and input $x \in \{0,1\}^{\ell_{in}}$

Output: 0 or 1

```
1: for $k = 1$ to $\ell_{out}$ do
2:     Compute $\mathbf{D}^{(k)} = \mathbf{A}^{(k)}_{1,0}\big(\prod_{i=1}^{L} \mathbf{C}^{(k)}_{i,x_{I(i)}}\big)$
3: end for
4: if $(\|\sum_{k=1}^{\ell_{out}} \mathbf{D}^{(k)}\|_\infty \leq \beta\,\ell_{out}\,(2m\beta)^{L-1})$ then
5:     return 1
6: else
7:     return 0
8: end if
```

5.2 Malicious Obfuscator for [WZ17]

We show a malicious obfuscator $\mathcal{A}_{CC}$ for the [WZ17] construction that violates correctness condition ϕ_2. We require $\mathcal{A}_{CC}$ to provide noticeable advantage in accepting a bad input, such that an efficient distinguisher, who is a priori informed of (f, α), cannot tell apart the honest and purported obfuscation instances.

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.

To design a malicious obfuscator, we sample an input $y \in \{0,1\}^{\ell_{in}}$. We think of y as a fixed backdoor that is used in all executions of the obfuscator. Let $\alpha' = f(y)$. 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 11.

Lemma 4. *The program output by Algorithm 11 accepts y and all correct inputs x. Algorithm 11 violates ϕ_2.*

Proof. To prove the above theorem, we show that $\mathcal{A}_{CC}$ violates ϕ_3, which implies violating correctness ϕ_2.

The fact that the [WZ17] scheme satisfies ϕ_3 means: for all $\lambda \in \mathbb{N}$, $f \in \mathcal{F}$, $\alpha \in \{0,1\}^{\ell_{out}(\lambda)}$ and $x \in \{0,1\}^{\ell_{in}}$, the following holds

$$\Pr_{\mathcal{O}_{CC}} [\, \mathcal{O}_{CC}(1^\lambda, (P^{(k)})_{k\in[\ell_{out}]}))(x) = CC[f, \alpha](x)\,] > 1 - \mu(\lambda)$$

where $\mu(\lambda)$ is a negligible function in λ. In particular, for the fixed input string $y \in \{0,1\}^{\ell_{in}}$ then we require for every $(P^{(k)})_{k\in[\ell_{out}]})$,

$$\Pr_{\mathcal{O}_{CC}} [\, \mathcal{O}_{CC}(1^\lambda, (P^{(k)})_{k\in[\ell_{out}]}))(y) = CC[f, \alpha](y)\,] > 1 - \mu(\lambda).$$

Let $CC[f, \alpha]$ be such that $CC[f, \alpha](y) = 0$. (This holds with overwhelming probability in λ.) The malicious obfuscator ensures that $\mathcal{A}_{CC}(1^\lambda, (P^{(k)})_{k\in[\ell_{out}]}), y)$ $(y) = 1$ with probability 1. Hence we have

$$\Pr_{\mathcal{A}_{CC}} [\, \mathcal{A}_{CC}(1^\lambda, (P^{(k)})_{k\in[\ell_{out}]}), y)(y) \neq CC[f, \alpha](y)\,] > 1 - \mu(\lambda)$$

where $\mu(\lambda)$ is negligible in λ. This proves that $\mathcal{A}_{CC}$ violates ϕ_3. □

Algorithm 11 Malicious Obfuscator $\mathcal{A}_{CC}\big(1^\lambda, (P^{(k)})_{k\in[\ell_{out}]}), y \in \{0,1\}^{\ell_{in}}\big)$

```
 1: if $(f(y) == \alpha)$ then
 2:     return $\mathcal{O}_{CC}(1^\lambda, (P^{(k)})_{k\in[\ell_{out}]}))$
 3: else
 4:     Set $\alpha' = f(y)$
 5:     Compute $z \xleftarrow{\$} \{1, \ldots, \ell_{out}\}$, s.t. $\alpha'_z \neq \alpha_z$
 6:     for $k = 1$ to $\ell_{out}$ do
 7:         Parse $P^{(k)} = (\pi^{(k)}_{i,b})_{i\in[L],b\in\{0,1\}}$
 8:         Sample $\mathbf{A}^{(k)}_{L+1,j} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$, $j \in \{0, \ldots, w-1\}$
 9:         if $(k == z)$ then
10:             $\mathbf{A}^{(z)}_{L+1,\alpha_z} = -\sum_{l\neq z} \mathbf{A}^{(l)}_{L+1,\alpha_l}$
11:             $\mathbf{A}^{(z)}_{L+1,\alpha'_z} = -\sum_{l\neq z} \mathbf{A}^{(l)}_{L+1,\alpha'_l}$
12:         end if
13:         Set $\mathbf{B}^{(k)}_{L+1}$ as in line 7 of Algorithm 9.
14:         Sample $(\mathbf{B}^{(k)}_i, t^{(k)}_i)$ as in lines 8-10 of Algorithm 9.
15:     end for
16:     Construct $\mathbf{C}^{(k)}_{i,j}$ as in lines 12-19 of Algorithm 9.
17:     return $\{\mathbf{A}^{(k)}_{1,0}\}_{k\in[\ell_{out}]}, \{(\mathbf{C}^{(k)}_{i,0}, \mathbf{C}^{(k)}_{i,1})\}_{k\in[\ell_{out}],i\in[L]}$
18: end if
```

5.3 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]. We remark that our proof follows an approach used in [WZ17] that ensures statistical indistinguishability, thus computational assumptions are needed for our result.

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 3. *Let $\mathcal{D} = \{\mathcal{D}_\lambda\}_{\lambda\in\mathbb{N}}$ be a family of distribution ensembles, such that $(C, \alpha) \leftarrow \mathcal{D}_\lambda$ satisfies $\mathbf{H}_{\mathsf{HILL}}(\alpha|f) \geq \gamma(\lambda, n)$, where λ is the security parameter,* *and ℓ_{in}, ℓ_{out}, L, w, n, m, q be polynomials in λ. Let (actually, it is the entropy to be bounded) $\ell_{out} = nmT \log q + \omega(\log \lambda)$. 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:*

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

Proof. The only difference between the honest and malicious obfuscator is line 7 of Algorithm 11. We have to 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 7 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 $(\mathbb{Z}_q^{n \times m})^T$, which has size q^{nmT}. This family is universal. Furthermore, the entropy of α is bounded as $\mathbf{H}_{\mathsf{HILL}}(\alpha'_1, \ldots, \alpha'_{\ell_{out}} | f) \geq nmT \log q + \omega(\log \lambda)$ and thus, by the leftover-hash lemma, the hash values (being the T maliciously formed matrices) are statistically indistinguishable from uniformly random. □

5.4 Auditable Obfuscator for [WZ17]

We now give the construction of our auditable obfuscator $\mathcal{AO}_{CC}(1^\lambda, (P^{(k)})_{k \in [\ell_{out}]})$ that provides security against $\mathcal{A}_{CC}(1^\lambda, (P^{(k)})_{k \in [\ell_{out}]}), y)$. In particular, we incorporate a verification procedure to ensure that the obfuscated program satisfies correctness ϕ_2. We follow the construction of Section 3, namely to de-randomise the obfuscation and to provide the seed for the PRG as the proof π that the obfuscation has been correctly followed. Recall that the malicious obfuscator (see Algorithm 11) samples the top-level matrices with an additional constraint $\sum_{k=1}^{\ell_{out}} \mathbf{A}^{(k)}_{n+1,\alpha'_k} = 0$ instead of sampling them uniformly at random from $\mathbb{Z}_q^{n \times m}$. This malicious behaviour is prevented by the auditable obfuscator enforcing that all matrices are chosen at random as required. Hence the auditor should definitely confirm the generation of all $\ell_{out} w - 1$ of the top level matrices. Applying Theorem 1, if the verify algorithm accepts then the obfuscated program has the intended correctness and no malicious behaviour has been introduced.

6 Other Obfuscation Schemes

We now show examples of randomised 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 that a small tweak gives perfect correctness and so a malicious backdoor cannot be inserted. We also briefly review the [BMZ19] scheme by Bartusek, Ma and Zhandry and show that the natural attempt to introduce a backdoor into their scheme is easily detected.

6.1 Reviewing the [GZ19] Construction

The hamming-ball membership obfuscator [GZ19] achieves DVBB security based on the *distributional decisional modular subset product* assumption. We write Δ for the hamming distance between binary strings.

Definition 9 (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*

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

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 10 (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*, denoted by $(\ell, r, B, D) - D_{MSP}$ states that the following distributions on $((p_i)_{i\in[\ell]}, q, A)$ are computationally indistinguishable: The first distribution samples $(\alpha_1, \ldots, \alpha_\ell) \xleftarrow{\$} D$, $(p_1, \ldots, p_\ell)$ a sequence of distinct primes sampled uniformly from $\{2, \ldots, B\}$, q a uniformly sampled safe prime in $\{B^r, \ldots, (1 + o(1))B^r\}$, and $A = \prod_{i=1}^{\ell} p_i^{\alpha_i}$* mod *$q$. The second distribution samples $(p_1, \ldots, p_\ell)$ 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 hamming-ball membership obfuscator works as follows: to encode α in a hamming ball $\mathsf{Ham}_{\alpha,r}$, sample distinct primes $(p_1, \ldots, p_\ell)$ uniformly at random in $\{2, \ldots, B\}$, together with a large prime modulus q (safe prime), such that $\prod_{i\in I} p_i < \frac{q}{2}$, for all $I \subset \{1, \ldots, \ell\}$, $|I| \leq r$. The final step is to 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 12.

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 Theorem 4, $\frac{s}{D}$ is a convergent of $\frac{E}{q}$ when $ND < \frac{q}{2}$ (which happens when $P_{\mathsf{Ham}_{\alpha,r}}(x) = 1$). The evaluation algorithm computes the set C of convergents of $\frac{E}{q}$. This enables extracting $\frac{s}{D}$, followed by recovering u_i, v_i and reconstructing the error vector $(e_i)_{i\in[\ell]}$. More specifically, for each $\frac{h}{k} \in C$, k and $kE \mod q$ are factored over $(p_1, \ldots, p_\ell)$ using Algorithm 13 to determine N and D that contains distinct primes from the sequence, subsequently flipping bits in $(e_i)_{i\in[\ell]}$. At the end, the algorithm compares $\mathcal{O}_{PT}(x \oplus e)$ with α'. If $x \notin \mathsf{Ham}_{\alpha,r}$, then with high probability the factors of N, D will not be unique and/or not contained in $(p_i)_{i\in[\ell]}$ and the algorithm correctly rejects the input. We give details in Algorithm 14.

Algorithm 12 Obfuscator $\mathcal{O}_H(1^\lambda, P, \mathcal{O}_{PT})$

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

Definition 11 (Continued Fractions and Convergents). *Let $z \in \mathbb{Q}$. The continued fraction representation of z is of the form*

$$z = a_0 + \cfrac{1}{a_1 + \cfrac{1}{\ddots + \cfrac{1}{a_n}}}$$

where $a_i \in \mathbb{N}^+$ for $i > 0$ and $a_0 \in \mathbb{N}$. The mth convergent of z, given by $\frac{h_m}{k_m}$, is defined as $h_m = a_m h_{m-1} + h_{m-2}$ $h_{-1} = 1, h_{-2} = 0$, $k_m = a_m k_{m-1} + k_{m-2}$ $k_{-1} = 0, k_{-2} = 1$. where $0 \leq m \leq n$, h_m, k_m are co-prime integers.

Theorem 4 (Diophantine Approximation). *Let $\beta \in \mathbb{R}$. Then, there exists $\frac{p}{q} \in \mathbb{Q}$, such that for $|\beta - \frac{p}{q}| < \frac{1}{2q^2}$, $\frac{p}{q}$ is a convergent of β.*

Galbraith and Zobernig are not precise about the correctness guarantees of their scheme. 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 (based on the choice of primes

$(p_1, \ldots, p_\ell)$). 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 11 of Algorithm 14: check that the Hamming weight of e is $\leq r$. This additional correctness check prevents malicious obduscation for the [GZ19] scheme.

Algorithm 13 Factor$((p_1, \ldots, p_\ell), a \in \mathbb{N})$

```
1: $F \leftarrow \{\ \}$;
2: for $i = 1$ to $\ell$ do
3:     if $p_i | a$ then
4:         $a \leftarrow a/p_i$
5:     end if
6: end for
7: if $(a == 1)$ then
8:     return $F$
9: else
10:    return $\perp$
11: end if
```

6.2 Reviewing the [BMZ19] Construction

We now discuss the non-malleable point-function obfuscation scheme by Bartusek, Ma and Zhandry [BMZ19] that follows correctness ϕ_3 and achieves DVBB in the generic group model. Let $\mathbb{G} = \langle g \rangle$ be a group of prime order $2^{\lambda-1} < q < 2^\lambda$, where λ is the security parameter. For a fixed $x \in \mathbb{Z}_q$, sample public values $a, b, c \overset{\$}{\leftarrow} \mathbb{Z}_q$ and publish $g^{P(x)}, g^{Q(x)}$, $g^{R(x)}$ 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 procedure simply calculates the three polynomials, maps them to the exponent space, and finally compares the values with the published obfuscation. Note that, 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 input y would imply the following holds: $P(x) = P(y)$, $Q(x) = Q(y)$ and $R(x) = R(y)$. This can be detected by a poly-time distinguisher $\mathcal{B}$ who knows x. By the same argument, there does not exist a malicious obfuscator for the point-function obfuscator by Fenteany and Fuller [FF20].

7 Conclusions

One of the open challenges in this area would be to develop more efficient verification algorithms, so that an auditor does not need to do roughly the same work as the obfuscator.

Algorithm 14 Evaluation Eval(with embedded values $1^\lambda, (p_i)_{i\in[\ell]}, q, A, \alpha', \mathcal{O}_{PT}$)

Input: $\ell = \ell(\lambda)$, $x \in \{0,1\}^\ell$
Output: 0 or 1

```
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:     $e \leftarrow (0,\ldots,0) \in \{0,1\}^\ell$
4:     $F \leftarrow \mathsf{Factor}((p_i)_{i\in[\ell]}, k)$; $F' \leftarrow \mathsf{Factor}((p_i)_{i\in[\ell]}, kE \mod q)$
5:     if $F \neq \perp$ and $F' \neq \perp$ then
6:         for $i = 1$ to $\ell$ do
7:             if $p_i \in F \cup F'$ then
8:                 $e_i \leftarrow 1$
9:             end if
10:        end for
11:        if $(\alpha' == \mathcal{O}_{PT}(x \oplus e))$ then
12:            return 1
13:        end if
14:    end if
15: end for
16: return 0
```

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.

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.

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*, pages 26–51. Springer, 2014.

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.

BGJS16. Saikrishna Badrinarayanan, Vipul Goyal, Aayush Jain, and Amit Sahai. Verifiable functional encryption. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016*, volume 10032 of *Lecture Notes in Computer Science*, pages 557–587, 2016.

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 *Annual International Cryptology Conference*, pages 731–752. Springer, 2018.

BLMZ19. James Bartusek, Tancrède Lepoint, Fermi Ma, and Mark Zhandry. New techniques for obfuscating conjunctions. In *Annual International Confer-*

ence on the Theory and Applications of Cryptographic Techniques, pages 636–666. Springer, 2019.

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.

BR17. Zvika Brakerski and Guy N Rothblum. Obfuscating conjunctions. *Journal of Cryptology*, 30(1):289–320, 2017.

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*, pages 731–758. Springer, 2022.

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.

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*, pages 73–90. Springer, 2009.

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.

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.

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

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.

GR07. Shafi Goldwasser and Guy N Rothblum. On best-possible obfuscation. In *Theory of Cryptography Conference*, pages 194–213. Springer, 2007.

GZ19. Steven D Galbraith and Lukas Zobernig. Obfuscated fuzzy hamming distance and conjunctions from subset product problems. In *Theory of Cryptography Conference*, pages 81–110. Springer, 2019.

HMLS07. Dennis Hofheinz, John Malone-Lee, and Martijn Stam. Obfuscation for cryptographic purposes. In *Theory of Cryptography Conference*, pages 214–232. Springer, 2007.

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 *Eurocrypt*, volume 7237, pages 700–718. Springer, 2012.

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

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.

ZGR19. Lukas Zobernig, Steven D Galbraith, and Giovanni Russello. When are opaque predicates useful? In *2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pages 168–175. IEEE, 2019.