Post-zeroizing Obfuscation: New Mathematical Tools, and the Case of Evasive Circuits

Saikrishna Badrinarayanan[1(✉)], Eric Miles[1], Amit Sahai[1], and Mark Zhandry[2,3]

[1] Center for Encrypted Functionalities, UCLA, Los Angeles, USA
{saikrishna,enmiles,sahai}@cs.ucla.edu
[2] MIT, Cambridge, USA
[3] Princeton University, Princeton, USA
mzhandry@princeton.edu

Abstract. Recent devastating attacks by Cheon et al. [Eurocrypt'15] and others have highlighted significant gaps in our intuition about security in candidate multilinear map schemes, and in candidate obfuscators that use them. The new attacks, and some that were previously known, are typically called "zeroizing" attacks because they all crucially rely on the ability of the adversary to create encodings of 0.

In this work, we initiate the study of *post-zeroizing obfuscation*, and we obtain a key new mathematical tool to analyze security in a post-zeroizing world. Our new mathematical tool allows for analyzing polynomials constructed by the adversary when given encodings of randomized matrices arising from a general matrix branching program. This technique shows that the types of encodings an adversary can create are much more restricted than was previously known, and is a crucial step toward achieving post-zeroizing security. We also believe the technique is of independent interest, as it yields efficiency improvements for existing schemes – efficiency improvements that have already found application in other settings.

Finally, we show how to apply our new mathematical tool to the special case of evasive functions. We show that our obfuscator survives *all known attacks* on the underlying multilinear maps, by proving that no top-level encodings of 0 can be created by a generic-model adversary. Previous obfuscators (for both evasive and general functions) were either analyzed in a less-conservative "pre-zeroizing" model that *does not* capture recent attacks, or were proved secure relative to assumptions that no longer have any plausible instantiation due to zeroizing attacks.

This paper subsumes a previous work of Sahai and Zhandry [35].

A. Sahai—Supported in part by a DARPA/ONR PROCEED award, NSF grants 1228984, 1136174, 1118096, 1065276,0916574 and 0830803, a Xerox Faculty Research Award, a Google Faculty Research Award, an equipment grant from Intel, and an Okawa Foundation Research Grant. This material is based upon work supported by the Defense Advanced Research Projects Agency through the U.S. Office of Naval Research under Contract N00014-11-1-0389. The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense, the National Science Foundation, or the U.S. Government.

M. Fischlin and J.-S. Coron (Eds.): EUROCRYPT 2016, Part II, LNCS 9666, pp. 764–791, 2016.
DOI: 10.1007/978-3-662-49896-5_27

1 Introduction

Over the past three years, *all* candidate constructions [16–18,22,27] of multilinear maps, also called graded encoding schemes, have been shown to suffer from "zeroizing" attacks [9,11,13–15,18,27,28,32] — and these attacks have in many cases been devastating.

Given this state of affairs, one would expect that the most-studied application of graded encodings schemes – indistinguishability obfuscation [7,20] – would be similarly devastated. However, quite surprisingly, until our work, *none* of the zeroizing attacks placed current obfuscation schemes over prime-order[1] graded encodings in jeopardy. In this paper, we ask: *Why is this the case?* Given the profound level of interest in obfuscation over the past two years [1,4,6,10,12,19,20,24,25,30,33,34,37], and given that so far all proposed obfuscation schemes rely on graded encoding schemes, we believe this question is of paramount importance. And indeed, before our work, no security analysis for obfuscation used a model or assumption that took into account the impact of zeroizing attacks.

Long-Term Vision. This paper seeks to initiate a research program whose aim is to build fully secure obfuscation schemes out of *weakened* graded encodings schemes – graded encoding schemes that are subject to zeroizing attacks. As the research cycle of construct-and-attack over the past 3 years has shown, building fully secure graded encoding schemes is a challenging task. Thus, our approach is to take a pessimistic view and see if, in fact, even weakened forms of graded encoding schemes suffice for constructing fully secure obfuscation. We note that even if future constructions of graded encoding schemes are successful in avoiding zeroizing attacks, the research program initiated by our work would still be valuable, because it will help to identify the minimal security properties actually needed by graded encoding schemes to achieve secure obfuscation. This could lead to greater efficiency.

The central contribution of our work is a new mathematical tool that characterizes when an adversary can set up the most basic requirement for a zeroizing attack, namely a top-level encoding of zero. Furthermore, we present this mathematical tool in a very general form, which even has consequences for efficiency of prime-order obfuscation constructions. We believe that our characterization lemma will prove valuable in the long-term study of both guiding research into new attacks on graded encodings as well as building secure obfuscation from weak graded encodings. We demonstrate this by applying our lemma to the case

[1] We note that certain *simplified* versions of obfuscation schemes over composite-order graded encoding schemes [37] have been broken by zeroizing attacks [15]. There are no published methods for converting the composite-order obfuscation schemes of [4,37] to the prime-order setting. Furthermore, zeroizing attacks over prime-order graded encoding schemes discovered prior to our work typically applied when the multiplicity of zero achieved at the top level is greater than zero, and if a prime-order conversion was attempted, care would need to be taken to ensure that such higher multiplicities do not occur.

of evasive circuits, for which we can show security even using only extremely weak graded encodings whose security completely breaks down when a top-level encoding of zero is found.

Background - Obfuscation. Obfuscation is a cryptographic tool that offers a powerful capability: software that can keep a secret. That is, consider a piece of software that makes use of a secret to perform its computation. Obfuscation allows us to transform this software so that it can be run *publicly*: anyone can obtain the full code of the program, run it, and see its outputs, but no one can learn anything about the embedded secret, beyond what can be learned by examining the outputs of the program.

The first candidate construction for a general-purpose obfuscator was given by Garg, Gentry, Halevi, Raykova, Sahai, and Waters [20]. This construction, and all subsequent works constructing candidate obfuscators [1,4,6,12,24,30,33,37], are built on top of another cryptographic primitive called a graded encoding scheme. In a graded encoding scheme, plaintext elements are encoded at various levels, and can be added and multiplied subject to algebraic restrictions relating to these levels. Further, there is a "top" level at which one can test whether an element encodes 0.

Background - Zeroizing Attacks. The many zeroizing attacks differ somewhat in their details, but each attack obeys the algebraic restrictions imposed by the graded encodings schemes, and critically they all share the need to create top-level 0-encodings. Indeed, many such encodings are needed for each attack, and the attacks require that these encodings have further structure.

Several of the works constructing candidate obfuscators prove security in an idealized "generic multilinear model" that seeks to capture the algebraic restrictions imposed by the graded encoding scheme candidates. However, the known zeroizing attacks use extra information that is provided by the zero-testing procedure, which is not captured in the standard generic model. Thus, a proof of security in the generic multilinear model by itself is no longer a persuasive argument of security. In particular, it is now crucial to gain a better understanding of exactly what types of top-level 0-encodings can be constructed in prime-order graded encoding schemes.

Our Contribution. We introduce a new mathematical tool for analyzing how an adversary can create 0-encodings given a set of randomized matrices. This tool both shows that the types of 0-encodings an adversary can create are much more restricted than was previously known, and that the adversary's behavior can be so controlled in a much richer set of circumstances than was previously known. We stress that this new tool, Theorem 4, was not present in any previous work, and allows for a much more fine-grained analysis of the adversary's behavior in prime-order settings than was previously available.

Briefly, we first consider an obfuscator $\mathcal{O}$ that can use much wider class of matrix branching programs than was previously known, most notably this class includes matrix branching programs that involve low-rank matrices. Theorem 4

shows that any polynomial p over (the encodings in) the obfuscation $\mathcal{O}(f)$ can be efficiently mapped to a poly-size set of inputs X such that p evaluates to an encoding of 0 if and only if *every* $x \in X$ satisfies $f(x) = 0$. For context, previous works both could not handle the case of low-rank matrices, and only gave a map that allowed the evaluation of p to be *simulated* given the set $\{f(x) \mid x \in X\}$, but did not show the stronger precise characterization of 0-encodings that we obtain. We stress that we do not know of any simpler way of obtaining such a characterization.

We now elaborate on Theorem 4, how it is proved, and how previous works that did not consider zeroizing attacks did not need and did not achieve such a theorem. Following that, we mention two applications of this new theorem, namely improving the efficiency of obfuscation, and obfuscating *evasive* functions in a model that captures all known attacks on graded encoding schemes.

1.1 Our Techniques

As stated above, the main technical challenge in our paper is to show that any polynomial p over the obfuscation $\mathcal{O}(f)$ can be efficiently mapped to a set of inputs X such that p evaluates to an encoding of 0 if and only if every $x \in X$ satisfies $f(x) = 0$.

One ingredient in our paper is the notion of strong straddling sets from [30], as this tool allows us to show that low-level encodings of 0 can be efficiently transformed into top-level encodings of 0. Thus, the only obstacle that remains is to prove Theorem 4 for top-level encodings.

The Technical Barrier – Kilian's Statistical Simulation. Before we proceed to provide intuition about our proof, let us consider the technical roots of how security was shown in previous works. In every paper constructing secure obfuscation for matrix branching programs so far [1,6,12,20,24,30,33] and in every different model that has been considered, one theorem has played a starring role in all security analyses: Kilian's statistical simulation theorem [29]. As relevant here, Kilian's theorem considers the setting where we randomize each matrix in a sequence of matrices as follows:

$$\widehat{\mathbf{B}_i} = \mathbf{R}_{i-1}^{-1}\mathbf{B}_i\mathbf{R}_i$$

where $\mathbf{R}_i$ are random invertible matrices for $i \in [\ell - 1]$, and identity otherwise. Note that this randomization does not affect the iterated product. Then, for any particular input x, if the iterated product is M, Kilian's theorem states that we can statistically simulate the collection of matrices $\{\widehat{\mathbf{B}_i}\}_{i\in[\ell]}$ knowing only M but with no knowledge of the original matrices $\{\mathbf{B}_i\}$.

Kilian's statistical simulation theorem has been a keystone in all previous analyses of obfuscation: in one way or another, all previous security analyses for obfuscation methods have found some way to isolate the adversary's view of the obfuscation to a single input. Once this isolation was accomplished, Kilian's theorem provided the assurance that the adversary's view of the obfuscation, as

it related to this single input, only encoded information about the output of the computation within M, and nothing more.

However, Kilian's statistical simulation theorem only allows for simulation. It does not rule out the possibility that an encoding of 0 may result no matter what the function outputs on the input in question. Indeed, it is not hard to construct an obfuscator that is secure in the generic model but allows for encodings of 0 even when the function being obfuscated always outputs 1. Moreover, Kilian's theorem only applies when the branching program matrices are full-rank. Indeed, if the matrices are allowed to be arbitrary rank, then it is *impossible* to simulate each of the matrices just given the product M, as there is no way to determine what the rank of each matrix should be, nor the ranks of various subproducts of matrices. (In the next subsection, we discuss the efficiency benefits of allowing low-rank matrices.) Because of this impossibility, we know of no way to generalize Kilian's theorem or its proof to obtain our theorem.

Our Approach. To obtain our result, we directly analyze what kinds of polynomials an adversary can generate using multilinear operations. We model the multilinear setting as follows. There is a universe set $[\ell]$, and for every subset $S \subseteq [\ell]$, we have a copy of $\mathbb{Z}_q$ that we name G_S. Then, the adversary has access to the following operations:

- Add: $G_S \times G_S \to G_S$, for every subset $S \subseteq [\ell]$.
- Mult: $G_S \times G_T \to G_{S \cup T}$, for every pair $S, T \subseteq [\ell] : S \cap T = \emptyset$.
- ZeroTest: $G_{[\ell]} \to \{\text{True}, \text{False}\}$.

This is sometimes called the "asymmetric" multilinear setting, is natively supported by known instantiations of prime-order graded encoding schemes [18], and was used in previous works. Observe that in this setting, if the adversary is given a matrix entirely encoded in $G_{\{1\}}$, then for example it is not possible for it to compute the rank of this matrix. This is because no two entries within this matrix can be multiplied together, since they both reside in the same group $G_{\{1\}}$, and multiplication is only possible across elements of groups corresponding to disjoint index sets.

Even though we do not rely on Kilian's simulation theorem, our obfuscator uses a matrix randomization scheme that is essentially[2] identical to the one used when applying Kilian's randomization. Our analysis then proceeds by considering the most general polynomial that the adversary can construct in $G_{[\ell]}$. More precisely, we consider every possible monomial m that can exist over the matrix entries that are given to the adversary, and we associate each such monomial m with a coefficient α_m that the adversary could potentially choose arbitrarily. Thus, the adversary's polynomial is a giant sum $p = \Sigma_m \alpha_m m$ over all these potential monomials.

[2] Because we consider rectangular matrices in general, we do need to modify this slightly. Also, for technical simplification, we consider the adjugate matrix rather than the inverse. However, for the purposes of this technical overview, these variations can be ignored.

Observe that the adversary can only extract useful information from this polynomial by passing it to ZeroTest, thereby determining if it is zero or not. However, recall that the randomizing matrices $\{\mathbf{R}_i\}$ are chosen uniformly during obfuscation. Therefore, by the Schwartz-Zippel lemma, we know that unless the adversary's polynomial p is the zero polynomial over the entries of the $\mathbf{R}_i$ matrices, ZeroTest will declare the polynomial to be nonzero with overwhelming probability. So, we restrict ourselves to analyzing adversarial polynomials that are identically zero over the entries of the $\mathbf{R}_i$ matrices.

Our new analysis differs at a fundamental level from Kilian's analysis. At the heart of the analysis is an argument based on the structure of random square matrices R and their inverses R^{-1} that allows us to argue about how terms that arise in R^{-1} can be cancelled using terms from R. In particular, we use the fact that:

$$R^{-1}_{i,\ell} = \frac{1}{\det(R)} \sum_{\sigma:\sigma(i)=\ell} \mathsf{sign}(\sigma) \left(\prod_{t \neq i} R_{\sigma(t),t} \right)$$

Our analysis is obtained by carefully considering different types of permutations σ that arise in the expression above, and how different permutations interfere with each other. (This exemplifies our conceptual departure from the proof of Kilian's theorem.) Our analysis shows that multilinear polynomials that allow for cancellation of R and R^{-1} terms are extremely constrained.

From this analysis, we conclude that any adversarial polynomial that is identically zero over the entries of the $\{\mathbf{R}_i\}$ matrices must in fact be the result of an honest iterated matrix multiplication (or a constant multiple thereof), which corresponds to evaluating $f(x)$ for some input x. In other words, such an adversarial polynomial will result in an encoding of 0 if and only if $f(x) = 0$, as desired. Even though the analysis as presented here is not efficient, we are still able to use it to yield an efficient simulator in our generic model. At a high level, this is done by using the Schwartz-Zippel lemma to "weed out" most adversarial polynomials without needing to examine their structure at all.

1.2 Applications

We now discuss two applications of our new analysis tool.

Efficiency of Obfuscation. Current techniques, while being asymptotically polynomial-time, lead to incredibly inefficient implementations of obfuscation. For example, the recent implementation of Apon et al. [2] for obfuscating (only) 16-bit point functions resulted in a 31 GB obfuscated program, which took over 6 h to generate and about 11 min to run on each input.

A major source of inefficiency is that the direct application of current obfuscators to circuits requires overhead that grows exponentially with the depth. This occurs because the level of multilinearity required grows exponentially with the depth, while current multilinear map candidates have complexity that grows polynomially with the level of multilinearity.

The work of Garg et al. [20] shows that, nevertheless, such a "core" obfuscator can be used to obfuscate general (high depth) circuits with a polynomial overhead through a "bootstrapping" procedure (see also [3,12,26]). However, bootstrapping based on existing core obfuscators entails overheads that are asymptotically polynomial but easily reach above 2^{100}. Such large overheads primarily arise due to the depth of the circuit processed by the core obfuscator (though, asymptotically, this circuit has depth logarithmic in the security parameter). Indeed, similarly large overheads arise when attempting to apply the core obfuscator to other programs represented in circuit form, since few interesting and non-learnable families of circuits have depth below, say, 50.

This suggests that practical implementations of obfuscation will only be able to handle functionalities that require a polynomial level of multilinearity, and not exponential. One such class of functionalities are those computable by small matrix branching programs, where evaluation corresponds to evaluating an iterated matrix product. This class of functionalities includes, among others, finite automata.

Unfortunately, natural representations of finite automata and other simple programs as branching programs require *low-rank* matrices. Though these representations can be made full rank by using much larger matrices, this results in substantial efficiency loss. The reason for this, intuitively, is that branching programs with invertible matrices model *reversible* computation, whereas general computation allows for previous states to be forgotten. While it is possible to convert an irreversible computation into a reversible one, the cost is a significant loss in efficiency. The ability to handle low-rank matrices is thus crucial to obtaining efficient obfuscators even for simple functionalities. As detailed in the preceding subsection, all previous constructions critically relied on full-rank matrices.

Armed with our new tool (Theorem 4), our construction *no longer requires full-rank matrices*, and even non-square matrices are allowed.[3] That is, we show for the first time how to obfuscate matrix branching programs that are represented with low-rank, rectangular matrices. This leads to more efficient obfuscators, even beyond previous works that lack a post-zeroizing proof of security; for details, see the full version of this paper. Our analysis also extends to other settings besides obfuscation: for example, Boneh et al. [8] rely on our analysis to obtain implementable constructions of order-revealing encryption.

Obfuscating Evasive Circuits in a Post-zeroizing Model. We view Theorem 4 as the first step on a path towards achieving obfuscation in a post-zeroizing world. As a "proof of concept" for this goal, we construct an obfuscator for a natural class of functions that, for the first time, is provably secure in a model that captures *all* known attacks on graded encoding schemes.

[3] We do require a mild natural technical condition, called *non-shortcutting*, on the branching program. Non-shortcutting can be achieved generically on any branching program with minimal overhead.

In previous works that prove security in a generic model, the graded encoding scheme's zero-test procedure is modeled as a Boolean function (i.e. one that returns a yes/no answer). In candidate constructions however, a successful zero-test actually returns an algebraic element in the ring of encodings, and this fact is crucially exploited in the zeroizing attacks. By contrast, our new model considers *any* encoding of 0 to be a complete break, thereby capturing these attacks.

We show how to obfuscate *evasive* functions [5] in this model, namely functions for which it is hard to find an input that evaluates to 0. (Typically one defines evasive functions as having hidden 1-outputs, but in terms of their functionality this is only a semantic difference.) A natural example of an evasive function is the "password check" function (typically called a point function), which evaluates to 0 on only a single, secret input. Obfuscating general evasive functions has many applications, including most notably obfuscating important classes of software patches that check for rare inputs on which the unpatched software is known to misbehave (see [5] for further discussion).

Prior to our work, except as a special case of general obfuscation, the only work that considered obfuscating general classes of evasive functions is that of [5]. However, the positive results in [5] were based on assumptions over approximate multilinear maps that are now known to be false when instantiated with current multilinear map candidates. Furthermore, the positive results in [5] did not consider completely arbitrary distributions of evasive circuits, as we do here.

Using our new analysis techniques, we prove the following.

Theorem 1 (informal). *There exists a PPT obfuscator $\mathcal{O}$ such that, for any evasive function family $\mathcal{C}$ on n-bit inputs and any efficient generic-model adversary $\mathcal{A}$,*

$$\Pr\left[\mathcal{A}(\mathcal{O}(C)) \text{ constructs an encoding of } 0\right] < \mathsf{negl}(n)$$

where the probability is over the choice of $C \leftarrow \mathcal{C}$ and the coins of $\mathcal{A}$ and $\mathcal{O}$.

Theorem 1 in particular implies the first *witness encryption* scheme [21] with a generic model proof that captures zeroizing attacks[4]. Indeed, in the original witness encryption protocol of [21] the attacker *can* produce top-level encodings of zero, and therefore the protocol is not secure in the post-zeroizing model. Subsequent witness encryption protocols [23,36] also allow top-level encodings of zero to be constructed.

In proving Theorem 1, we show that the "bootstrapping" theorem of [20] extends to the setting of evasive functions. (As mentioned above, this theorem transforms a core obfuscator for a "small" class of functions into an obfuscator for all efficient functions.) We observe that the proof of this theorem only uses the core obfuscator on evasive functions, and we show that it holds only assuming the core obfuscator's security on such functions. In particular, we show that

[4] When building witness encryption from obfuscation, witness encryption security only requires the obfuscator to be secure when obfuscating functions that always evaluate to 0, which are in particular evasive.

Theorem 1 applies to all evasive functions and not only those on which the core obfuscator operates. Interestingly, the more recent bootstrapping technique of Applebaum [3] cannot be used for our purposes, because it inherently produces encodings of 0 regardless of the function being obfuscated.

Directions for Future Work. The obvious next step is to consider obfuscating non-evasive functions. To do so, we will need to look precisely at the kinds of post-zero-test information that can be obtained using zeroizing attacks during zero testing for general (non-evasive) functions. We note that our paper answers a critical first question toward this goal: we show that in our scheme, the *only* way that the adversary can create top-level encodings of zero are the prescribed ways of evaluating the function at a particular input. This is a necessary first step in understanding what kinds of information can arise in the general case, and whether this information can lead to more sophisticated attacks.

Subsequent work by a subset of the authors [31] has shown how to attack candidate iO schemes (including the one here), when implemented with the [18] multilinear map candidate, by further analyzing the polynomials that correspond to honest evaluations of the obfuscated function. However, we remark that this attack still crucially relies on encodings of 0 (corresponding to 0-outputs of the function), and as a result it cannot be mounted when the function being obfuscated is evasive.

Organization. In Sect. 2 we give some preliminary definitions and background information. In Sect. 3 we define our obfuscator for matrix branching programs. In Sect. 4 we prove the key technical theorem that analyzes adversarially-constructed polynomials over the obfuscation. The proof of VBB security is outlined in Sect. 5 (due to space limitations, the complete proof is deferred to the full version of this paper). In Sect. 6 we prove that, when obfuscating evasive functions, no encodings of zero can be created.

2 Preliminaries

2.1 Evasive Circuits

We define evasive circuit collections as in Barak et al. [5], except that in our definition it is hard to find a 0-output (typically one says that it is hard to find a 1-output).

Definition 1. *A function family* $\{\mathcal{C}_\ell\}_{\ell\in\mathbb{N}}$ *is* evasive *if for every oracle-aided adversary* $\mathcal{A}^{(\cdot)}$ *that makes at most* $\mathsf{poly}(\ell)$ *queries on input* 1^ℓ*, and every* $\ell \in \mathbb{N}$*:*

$$\Pr_{C\leftarrow\mathcal{C}_\ell}\left[C\left(\mathcal{A}^C\left(1^\ell\right)\right)=0\right]=\mathsf{negl}(\ell).$$

$\{\mathcal{C}_\ell\}_{\ell\in\mathbb{N}}$ *is* evasive with auxiliary input Aux *for a (possibly randomized) function* $\mathsf{Aux}:\mathcal{C}_\ell\rightarrow\{0,1\}^*$ *if* $\mathcal{A}$ *additionally receives* $\mathsf{Aux}(C)$ *when its oracle is* C*.*

2.2 Obfuscation

We now give the definition of virtual black-box obfuscation in an idealized model, identical to the model studied in Barak et al. [6] and Ananth et al. [1], with one exception: we also consider giving both the adversary and simulator an auxiliary input determined by the program.

Definition 2 (Virtual Black-Box Obfuscation in an $\mathcal{M}$-idealized model). *For a (possibly randomized) oracle $\mathcal{M}$, a circuit class $\{\mathcal{C}_\ell\}_{\ell\in\mathbb{N}}$, and an efficiently computable deterministic function $\mathsf{Aux}_\ell : \mathcal{C}_\ell \to \{0,1\}^{t_\ell}$, we say that a uniform PPT oracle machine $\mathcal{O}$ is a "Virtual Black-Box" Obfuscator for $\{\mathcal{C}_\ell\}_{\ell\in\mathbb{N}}$ in the $\mathcal{M}$-idealized model with respect to auxiliary information Aux_ℓ, if the following conditions are satisfied:*

- *Functionality: For every $\ell \in \mathbb{N}$, every $C \in \mathcal{C}_\ell$, every input x to C, and for every possible coins for $\mathcal{M}$:*

$$\Pr[(\mathcal{O}^{\mathcal{M}}(C))(x) \neq C(x)] \leq \mathrm{negl}(|C|) \ ,$$

  where the probability is over the coins of $\mathcal{C}$.
- *Polynomial Slowdown: there exist a polynomial p such that for every $\ell \in \mathbb{N}$ and every $C \in \mathcal{C}_\ell$, we have that $|\mathcal{O}^{\mathcal{M}}(C)| \leq p(|C|)$.*
- *Virtual Black-Box: for every PPT adversary $\mathcal{A}$ there exist a PPT simulator Sim, and a negligible function μ such that for all PPT distinguishers D, for every $\ell \in \mathbb{N}$ and every $C \in \mathcal{C}_\ell$:*

$$\left|\Pr\left[D\left(\mathcal{A}^{\mathcal{M}}\left(\mathcal{O}^{\mathcal{M}}(C), \mathsf{Aux}_\ell(C)\right)\right) = 1,\right] - \Pr\left[D\left(\mathsf{Sim}^{C}\left(1^{|C|}, \mathsf{Aux}_\ell(C)\right)\right) = 1\right]\right| \leq \mu(|C|) \ ,$$

  where the probabilities are over the coins of D, $\mathcal{A}$, Sim, $\mathcal{O}$ and $\mathcal{M}$.

Note that in this model, both the obfuscator and the evaluator have access to the oracle $\mathcal{M}$ but the function family that is being obfuscated does not have access to $\mathcal{M}$.

We also define the average-case version of VBB obfuscation, which is the correct security notion when obfuscating evasive circuit collections.

Definition 3 (Average-case Virtual Black-Box Obfuscation in an $\mathcal{M}$-idealized model). *Let $\mathcal{M}$, $\{\mathcal{C}_\ell\}_{\ell\in\mathbb{N}}$, and Aux_ℓ be as in Definition 2. We say that a uniform PPT oracle machine $\mathcal{O}$ is an average-case Virtual Black-Box Obfuscator for $\{\mathcal{C}_\ell\}_{\ell\in\mathbb{N}}$ in the $\mathcal{M}$-idealized model with respect to auxiliary information Aux_ℓ, if it satisfies all properties in Definition 2 except that in the Virtual Black-Box property the probabilities are over a uniform choice of $C \leftarrow \mathcal{C}_\ell$ (as opposed to $\forall C \in \mathcal{C}_\ell$).*

Definition 4 (Average-case Indistinguishability Obfuscation in an $\mathcal{M}$-idealized model). *For a (possibly randomized) oracle $\mathcal{M}$, a circuit class $\{\mathcal{C}_\ell\}_{\ell\in\mathbb{N}}$, we say that a uniform PPT oracle machine $\mathcal{O}$ is an Average-case Indistinguishability Obfuscator for $\{\mathcal{C}_\ell\}_{\ell\in\mathbb{N}}$ in the $\mathcal{M}$-idealized model if the following conditions are satisfied:*

- *Functionality:* *Same as in the definition of VBB.*
- *Polynomial Slowdown:* *Same as in the definition of VBB.*
- *Indistinguishability:* *For every PPT Distinguisher D, there exists a negligible function μ such that the following holds: for every $\ell \in \mathbb{N}$, for a uniform choice of circuit $C \in \mathcal{C}_\ell$ and for every pair of circuits $C_0, C_1 \in \mathcal{C}_\ell$ that compute the same function as C, we have:*

$$\left|\Pr\left[D(\mathcal{O}^{\mathcal{M}}(C_0)) = 1\right] - \Pr\left[D(\mathcal{O}^{\mathcal{M}}(C_1)) = 1\right]\right| \leq \mu(|C|) \ ,$$

where the probabilities are over the coins of D, $\mathcal{O}$, $\mathcal{M}$ and the choice of C.

Note that in this model, both the obfuscator and the evaluator have access to the oracle $\mathcal{M}$ but the function family that is being obfuscated does not have access to $\mathcal{M}$.

2.3 Branching Programs

Here we define the main type of branching program we consider. A detailed description of other types of branching programs, and how to build these branching programs from other computational models, can be found in the full version of this paper.

Definition 5. *A* dual-input generalized matrix branching program *of length ℓ and shape $(d_0, d_1, \ldots, d_\ell) \in (\mathbb{Z}^+)^{\ell+1}$ for n-bit inputs is given by a sequence*

$$BP = \left(\mathsf{inp}_0, \mathsf{inp}_1, \{\mathbf{B}_{i,b_0,b_1}\}_{i\in[\ell],b_0,b_1\in\{0,1\}}\right)$$

where $\mathbf{B}_{i,b_0,b_1} \in \mathbb{Z}^{d_{i-1}\times d_i}$ are $d_{i-1} \times d_i$ matrices, and $\mathsf{inp} : [\ell] \to [n]$ is the evaluation function of BP. BP defines the following three functions:

- $BP_{arith} : \{0,1\}^n \to \mathbb{Z}^{d_0\times d_\ell}$ *computed as* $BP_{arith}(x) = \prod_{i=1}^{n} \mathbf{B}_{i,x_{\mathsf{inp}_0(i)},x_{\mathsf{inp}_1(i)}}$
- $BP_{bool} : \{0,1\}^n \to \{0,1\}^{d_0\times d_\ell}$ *computed as* $BP_{bool}(x)_{j,k} = \begin{cases} 0 & \text{if } BP_{arith}(x)_{j,k} = 0 \\ 1 & \text{if } BP_{arith}(x)_{j,k} \neq 0 \end{cases}$
- $BP_{bool(q)} : \{0,1\}^n \to \{0,1\}^{d_0\times d_\ell}$ *computed as* $BP_{bool(q)}(x)_{j,k} = \begin{cases} 0 & \text{if } BP_{arith}(x)_{j,k} = 0 \mod q \\ 1 & \text{if } BP_{arith}(x)_{j,k} \neq 0 \mod q \end{cases}$

A matrix branching program is t-bounded if $|BP_{arith}(x)_{j,k}| \leq t$ for all x, j, k.

Next, we define a notion of non-shortcuttingfor matrix branching programs, which roughly states that it is not possible to determine any of the output components of $BP_{arith/bool}$ without carrying out the entire matrix product. In the

case $d_0 = d_\ell = 1$ (so that the branching program outputs just a single element), this translates to requiring that no strict sub-product $\prod_{i=i_0}^{i_1} \mathbf{B}_{i,x_{\mathsf{inp}_0(i)},x_{\mathsf{inp}_1(i)}}$ for $(i_0, i_1) \neq (1, n)$ of the overall matrix product evaluates to an all-zero matrix. Clearly, if some sub-product evaluates to zero, the entire product would evaluate to zero, and so the evaluation could stop after computing just the sub-product. We call this a short-cut, and non-shortcuttingis the requirement that there are no shortcuts for any inputs. In the more general case of arbitrary d_0, d_ℓ, the condition becomes slightly more technical, and is given below:

Definition 6. *A dual-input generalized matrix branching program is* non-shortcutting *if, for any input x, and any $j \in [d_0]$ and any $k \in [d_\ell]$, the following holds:*

$$\mathbf{e}_j^T \cdot \left(\prod_{i=1}^{\ell-1} \mathbf{B}_{i,x_{\mathsf{inp}_0(i)},x_{\mathsf{inp}_1(i)}} \right) \neq 0^{d_{\ell-1}} \quad \text{and} \quad \left(\prod_{i=2}^{\ell} \mathbf{B}_{i,x_{\mathsf{inp}_0(i)},x_{\mathsf{inp}_1(i)}} \right) \cdot \mathbf{e}_k \neq 0^{d_1}$$

where $\mathbf{e}_j$ and $\mathbf{e}_k$ are the jth and kth standard basis vectors of the correct dimension. Equivalently, each row of the product $\prod_{i=1}^{\ell-1} \mathbf{B}_{i,x_{\mathsf{inp}_0(i)},x_{\mathsf{inp}_1(i)}}$ and each column of the product $\prod_{i=2}^{\ell} \mathbf{B}_{i,x_{\mathsf{inp}_0(i)},x_{\mathsf{inp}_1(i)}}$ has at least one non-zero entry.

Matrix Branching Program Samplers. We now define a matrix branching program sampler (MBPS). Roughly, an MBPS is a procedure that takes as input a modulus q, and outputs a matrix branching program BP. However, we will be interested mainly in the function $BP_{bool(q)}$.

Definition 7. *A matrix branching program sampler (MBPS) is a possibly randomized procedure BP^S that takes as input a modulus q satisfying $q > t$ for some bound t. It outputs a matrix branching program.*

Fact 2. *Any matrix branching program BP with bound t can trivially be converted into a matrix branching program sampler BP^S with the same bound t, such that if $BP' \leftarrow BP^S(q)$, then $BP'_{bool(q)}(x) = BP_{bool}(x)$.*

2.4 The Ideal Graded Encoding Model

In this section, we describe the ideal graded encoding model. This section has been taken almost verbatim from [1,6]. All parties have access to an oracle $\mathcal{M}$, implementing an ideal graded encoding. The oracle $\mathcal{M}$ implements an idealized and simplified version of the graded encoding schemes from [18]. The parties are provided with encodings of various elements at different levels. They are allowed to perform arithmetic operations of addition/multiplication and testing equality to zero as long as they respect the constraints of the multilinear setting. We start by defining an algebra over the elements.

Definition 8. *Given a ring R and a universe set $\mathbb{U}$, an element is a pair (α, S) where $\alpha \in R$ is the* value *of the element and $S \subseteq \mathbb{U}$ is the* index *of the element. Given an element e we denote by $\alpha(e)$ the value of the element, and we denote by $S(e)$ the index of the element. We also define the following binary operations over elements:*

- *For two elements* e_1, e_2 *such that* $S(e_1) = S(e_2)$*, we define* $e_1 + e_2$ *to be the element* $(\alpha(e_1) + \alpha(e_2), S(e_1))$*, and* $e_1 - e_2$ *to be the element* $(\alpha(e_1) - \alpha(e_2), S(e_1))$*.*
- *For two elements* e_1, e_2 *such that* $S(e_1) \cap S(e_2) = \emptyset$*, we define* $e_1 \cdot e_2$ *to be the element* $(\alpha(e_1) \cdot \alpha(e_2), S(e_1) \cup S(e_2))$*.*

We will often use the notation $[\alpha]_S$ to denote the element (α, S). Next, we describe the oracle $\mathcal{M}$. $\mathcal{M}$ is a stateful oracle mapping elements to "generic" representations called *handles*. Given handles to elements, $\mathcal{M}$ allows the user to perform operations on the elements. $\mathcal{M}$ will implement the following interfaces:

Initialization. $\mathcal{M}$ will be initialized with a ring R, a universe set $\mathbb{U}$, and a list L of initial elements. For every element $e \in L$, $\mathcal{M}$ generates a handle. We do not specify how the handles are generated, but only require that the value of the handles are independent of the elements being encoded, and that the handles are distinct (even if L contains the same element twice). $\mathcal{M}$ maintains a handle table where it saves the mapping from elements to handles. $\mathcal{M}$ outputs the handles generated for all the elements in L. After $\mathcal{M}$ has been initialized, all subsequent calls to the initialization interface fail.

Algebraic Operations. Given two input handles h_1, h_2 and an operation $\circ \in \{+, -, \cdot\}$, $\mathcal{M}$ first locates the relevant elements e_1, e_2 in the handle table. If any of the input handles does not appear in the handle table (that is, if the handle was not previously generated by $\mathcal{M}$) the call to $\mathcal{M}$ fails. If the expression $e_1 \circ e_2$ is undefined (i.e., $S(e_1) \neq S(e_2)$ for $\circ \in \{+, -\}$, or $S(e_1) \cap S(e_2) \neq \emptyset$ for $\circ \in \{\cdot\}$) the call fails. Otherwise, $\mathcal{M}$ generates a new handle for $e_1 \circ e_2$, saves this element and the new handle in the handle table, and returns the new handle.

Zero Testing. Given an input handle h, $\mathcal{M}$ first locates the relevant element e in the handle table. If h does not appear in the handle table (that is, if h was not previously generated by $\mathcal{M}$) the call to $\mathcal{M}$ fails. If $S(e) \neq \mathbb{U}$, the call fails. Otherwise, $\mathcal{M}$ returns 1 if $\alpha(e) = 0$, and returns 0 if $\alpha(e) \neq 0$.

2.5 Straddling Set Systems

We use the *strong* straddling set system of [30], which modifies the straddling set system of [6] to obtain a denser intersection graph between the subsets. This extra power is used in Sect. 6 when showing that the adversary cannot create low-level encodings of 0.

Definition 9 (Strong straddling set system). *A* strong straddling set system *with* n *entries is a collection of sets* $\mathbb{S} = \{S_{i,b} : i \in [n], b \in \{0,1\}\}$ *over a universe* $\mathbb{U}$*, such that* $\cup_{i\in[n]} S_{i,0} = \mathbb{U} = \cup_{i\in[n]} S_{i,1}$*, and the following holds.*

- *(Collision at universe.) If* $C, D \subseteq \mathbb{S}$ *are distinct non-empty collections of disjoint sets such that* $\bigcup_{S\in C} S = \bigcup_{S\in D} S$*, then* $\exists b \in \{0,1\}$ *such that* $C = \{S_{i,b}\}_{i\in[n]}$ *and* $D = \{S_{i,1-b}\}_{i\in[n]}$*.*

– *(Strong intersection.) For every* $i, j \in [n]$, $S_{i,0} \cap S_{j,1} \neq \emptyset$.

We will need the following simple lemma.

Lemma 1. *Let* $\mathbb{S} = \{S_{i,b} : i \in [n], b \in \{0,1\}\}$ *be a strong straddling set system over a universe* $\mathbb{U}$*. Then for any* $T \subsetneq \mathbb{U}$ *that can be written as a disjoint union of sets from* $\mathbb{S}$*, there is a unique* $b \in \{0,1\}$ *such that* $T = \bigcup_{i \in I} S_{b,i}$ *for some* $I \subseteq [n]$*.*

Proof. By the second property of Definition 9, any pairwise disjoint collection of sets from $\mathbb{S}$ must be either all of the form $S_{i,0}$ or all of the form $S_{i,1}$. If there are two sets $I_0, I_1 \subseteq [n]$ such that $\bigcup_{i \in I_0} S_{i,0} = T = \bigcup_{i \in I_1} S_{i,1}$, then by the first property of Definition 9 we must have $T = \mathbb{U}$ which contradicts our assumption.

We use the following construction from [30].

Construction 3 (Strong straddling set system). *Define* $\mathbb{S} = \{S_{i,b} : i \in [n], b \in \{0,1\}\}$ *over a universe* $\mathbb{U} = \{1, 2, ..., n^2\}$ *as follows for all* $1 \leq i \leq n$*.*

$$S_{i,0} = \{n(i-1)+1, n(i-1)+2, \ldots, ni\} \qquad S_{i,1} = \{i, n+i, 2n+i, \ldots, n(n-1)+i\}$$

3 Obfuscator for Low-Rank Branching Programs

We now describe our obfuscator for generalized matrix branching programs. Our obfuscator is essentially the same as the obfuscator of Ananth et al. [1]. The differences are as follows:

– We view branching programs as including the bookends. While the bookends of previous works did not depend on the input, they can in our obfuscator. However, for [1], this distinction is superficial: the bookends of [1] can be "absorbed" into the branching program by merging them with the left-most and right-most matrices of the branching program. This does not change functionality, since this merging always happens during evaluation, and it does not change security, since the adversary can perform the merging himself.
– We allow our branching program to have singular and rectangular matrices. We do, however, require the branching program to be non-shortcutting. Note that a branching program with square invertible internal matrices and non-zero bookend vectors, such as in [1], necessarily is non-shortcutting.
– We allow branching programs to output multiple bits — that is, the function computed by our obfuscated program will be BP_{bool}, which is a matrix of 0/1 entries. In order to prove security, we will have to perform additional randomization. However, in the case of single-bit outputs, this additional randomization is redundant.

Input. The input to our obfuscator is a dual-input matrix branching program sampler BP^S of length ℓ, shape $(d_0, d_1, \ldots, d_\ell)$, and bound t. The first step is to choose a large prime q for the graded encodings. Then sample $BP \leftarrow BP^S(q)$. Write

$$BP = (\mathsf{inp}_0, \mathsf{inp}_1, \{\mathbf{B}_{i,b_0,b_1}\})$$

We require BP^S to output BP satisfying the following properties:

- BP is non-shortcutting.
- For each i, $\mathsf{inp}_0(i) \neq \mathsf{inp}_1(i)$
- For each pair $(j,k) \in [n]^2$, there exists an $i \in [\ell]$ such that $(\mathsf{inp}_0(i), \mathsf{inp}_1(i)) = (j,k)$ or $(\mathsf{inp}_1(i), \mathsf{inp}_0(i)) = (j,k)$

For ease of notation in our security proof, we will also assume that each input bit is used exactly m times, for some integer m. In other words, for each $i \in [n]$, the sets $\mathsf{ind}(i) = \{j : \mathsf{inp}_b(j) = i \text{ for some } b \in \{0,1\}\}$ have the same size. This requirement, however, is not necessary for security.

Step 1: Randomize BP. First, similar to previous works, we use Kilian [29] to randomize BP, obtaining a randomized branching program BP'. This is done as follows.

- Let q be a sufficiently large prime of $\Omega(\lambda)$ bits.
- For each $i \in [\ell - 1]$, choose a random matrix $\mathbf{R}_i \in \mathbb{Z}_q^{d_i \times d_i}$. Set $\mathbf{R}_0, \mathbf{R}_\ell$ to be identity matrices of the appropriate size. Define

$$\widehat{\mathbf{B}_{i,b_0,b_1}} = \mathbf{R}_{i-1}^{adj} \cdot \mathbf{B}_{i,b_0,b_1} \cdot \mathbf{R}_i$$

- For each $s \in [d_0]$, choose a random β_s and set $\mathbf{S}$ to be the $d_0 \times d_0$ diagonal matrix with the β_s along the diagonal. For each $t \in [d_\ell]$, choose a random γ_t and set $\mathbf{T}$ to be the $d_\ell \times d_\ell$ diagonal matrix with γ_t along the diagonal. Set

$$\mathbf{C}_{1,b_0,b_1} = \mathbf{S} \cdot \widehat{\mathbf{B}_{1,b_0,b_1}} \quad \mathbf{C}_{\ell,b_0,b_1} = \widehat{\mathbf{B}_{1,b_0,b_1}} \cdot \mathbf{T} \quad \mathbf{C}_{i,b_0,b_1} = \widehat{\mathbf{B}_{i,b_0,b_1}} \text{ for each } i \in [2, \ell-1]$$

  We note that this additional randomization step is not present in previous works, but is required to handle multi-bit outputs
- For each $i \in [\ell]$, $b_0, b_1 \in \{0,1\}$, choose a random $\alpha_{i,b_0,b_1} \in \mathbb{Z}_p$, and define

$$\mathbf{D}_{i,b_0,b_1} = \alpha_{i,b_0,b_1} \mathbf{C}_{i,b_0,b_1}$$

Then define $BP' = (\mathsf{inp}_0, \mathsf{inp}_1, \{\mathbf{D}_{i,b_0,b_1}\})$. Observe that $BP'_{bool(q)}(x) = BP_{bool(q)}(x)$ for all x.

Step 2: Create Set Systems. Consider a universe $\mathbb{U}$, and a partition $\mathbb{U}_1, \ldots, \mathbb{U}_\ell$ of $\mathbb{U}$ into equal sized disjoint sets: $|\mathbb{U}_i| = 2m - 1$. Let $\mathbb{S}^j$ be a straddling set system over the elements of $\mathbb{U}_j$. Note that $\mathbb{S}^j$ will have m entries, corresponding to the number of times each input bit is used. We now associate the elements of $\mathbb{S}_j$ to the indicies of BP that depend on x_j:

$$\mathbb{S}^j = \{S^j_{k,b} : k \in \mathsf{ind}(j), b \in \{0,1\}\}$$

Next, we associate a set to each element output by the randomization step. Recall that in a dual-input relaxed matrix branching program, each step depends on two fixed bits in the input defined by the evaluation functions inp_0 and inp_1. For each step $i \in [n], b_0, b_1 \in \{0,1\}$, we define the set $S(i, b_0, b_1)$ using the straddling sets for input bits $inp_1(i)$ and $inp_2(i)$ as follows:

$$S_{i,b_0,b_1} = S^{\mathsf{inp}_0(i)}_{i,b_0} \cup S^{\mathsf{inp}_1(i)}_{i,b_1}$$

Step 3: Initialization. $\mathcal{O}$ initializes the oracle $\mathcal{M}$ with the ring $\mathbb{Z}_p$ and the universe $\mathbb{U}$. Then it asks for the encodings of the following elements:

$$\{(\mathbf{D}_{i,b_0,b_1}[j,k], S_{i,b_0,b_1})\}_{i\in[\ell],b_0,b_1\in\{0,1\},j\in[d_{i-1}],k\in[d_i]}$$

$\mathcal{O}$ receives a list of handles back from $\mathcal{M}$. Let $[\beta]_S$ denote the handle for (β, S), and for a matrix M, let $[M]_S$ denote the matrix of handles $([M]_S)[j,k] = [M[j,k]]_S$. Thus, $\mathcal{O}$ receives the handles:

$$\left\{[\mathbf{D}_{i,b_0,b_1}]_{S_{i,b_0,b_1}}\right\}_{i\in[\ell],b_0,b_1\in\{0,1\}}$$

Output. $\mathcal{O}(BP^S)$ outputs these handles, along with the length ℓ, shape $d_0, \ldots, d_\ell$, and input functions $\mathsf{inp}_0, \mathsf{inp}_1$, as the obfuscated program. Denote the resulting obfuscated branching program as $BP^{\mathcal{O}}$

Evaluation. To evaluate $BP^{\mathcal{O}}$ on input x, use the oracle $\mathcal{M}$ to add and multiply encodings in order to compute the product

$$h = \left[\prod_{i\in[\ell]} \mathbf{D}_{i,x_{\mathsf{inp}_0(i)},x_{\mathsf{inp}_1(i)}}\right]_{\mathbb{U}} = \prod_{i\in[\ell]} \left[\mathbf{D}_{i,x_{\mathsf{inp}_0(i)},x_{\mathsf{inp}_1(i)}}\right]_{S_{i,x_{\mathsf{inp}_0(i)},x_{\mathsf{inp}_1(i)}}}$$

h is a $d_0 \times d_\ell$ matrix of encodings relative to $\mathbb{U}$. Next, use $\mathcal{M}$ to test each of the components of h for zero, obtaining a matrix $h_{bool} \in \{0,1\}^{d_0\times d_\ell}$. That is, if the zero test on returns a 1 on $h[s,t]$, $h_{bool}[s,t]$ is 0, and if the zero test returns a 0, $h_{bool}[s,t]$ is 1.

Correctness of Evaluation. The following shows that all calls to the oracle $\mathcal{M}$ succeed:

Lemma 2 (Adapted from [1]). *All calls made to the oracle $\mathcal{M}$ during obfuscation and evaluation succeed.*

It remains to show that the obfuscated program computes the correct function. Fix an input x, and define $b^i_c = x_{\mathsf{inp}_c(i)}$ for $i \in [\ell], c \in \{0,1\}$. From the description above, $BP^{\mathcal{O}}$ outputs 0 at position $[s,t]$ if and only if

$$0 = \left(\prod_{i\in[\ell]} \mathbf{D}_{i,b^i_0,b^i_1}\right)[s,t] = \beta_s\gamma_t \left(\prod_{i\in[\ell]} \alpha_{i,b^i_0,b^i_1}\mathbf{R}^{adj}_{i-1}\cdot\mathbf{B}_{i,b^i_0,b^i_1}\cdot\mathbf{R}_i\right)[s,t]$$

$$= \beta_s\gamma_t \left(\left(\prod_{i\in[\ell]} \alpha_{i,b^i_0,b^i_1}\right)\left(\prod_{i\in[\ell]} \mathbf{B}_{i,b^i_0,b^i_1}\right)\right)[s,t] = \left(\beta_s\gamma_t \prod_{i\in[\ell]} \alpha_{i,b^i_0,b^i_1}\right)(BP_{arith}(x)[s,t])$$

With high probability $\beta_s, \gamma_t, \alpha_{i,b_0,b_1} \neq 0$, meaning $BP_{arith}(x)[s,t] = 0 \mod q$ if and only if the zero test procedure on position $[s,t]$ gives 0. Therefore, $BP^{\mathcal{O}}(x) = BP_{bool(q)}(x)$ for the branching program BP sampled from BP^S.

4 Polynomials on Kilian-Randomized Matrices

In this section, we prove a theorem about polynomials on the Kilian-randomized matrices from the previous section. Our high level goal is to show polynomials the adversary tries to construct other than the correct matrix products will be useless to the adversary. In this section, we focus on a simpler case where the polynomial is only over matrices corresponding to a single input. In the following section, we use the results of this section to prove the general case.

Previous works showed the single-input case using Kilian simulation [6,12], or a variant of it [1,33]. Namely, these works queried the function oracle to determine what the result of the matrix product $P(x)$ should be. Then, they tested the polynomial on random matrices, subject to the requirement that the product equaled $P(x)$, to see what the result was. Unfortunately, this step of the analysis does indicate what the outputs of the polynomial may be, only that they can be simulated. If the polynomial were to output zero, this would correspond to the adversary obtaining a zero encoding, which would violate security in our post-zeroizing model.

Moreover, previous works crucially relied on the fact that the matrices the polynomial is tested on come from the same distribution as the matrices would in the branching program. This requires the branching program to consist of square invertible matrices. However, we need to be able to handle generalized matrix branching programs with rectangular and low-rank matrices.

In light of the two issues above, we need to replace the Kilian randomization theorem with a new theorem suitable in our setting.

Let $d_1, \ldots, d_{n-1}$ be positive integers and $d_0 = d_n = 1$. Let $\widehat{\mathbf{A}_k}$ for $k \in [n]$ be $d_{k-1} \times d_k$ matrices of variables.

Definition 10. *Let $d_k, \widehat{\mathbf{A}_k}$ be as above. Consider a multilinear polynomial p on the variables in $\{\widehat{\mathbf{A}_k}\}_{k\in[n]}$. We call p* allowable *if each monomial in the expansion of p contains at most one variable from each of the $\widehat{\mathbf{A}_k}$.*

As an example of an allowable polynomial, consider the *matrix product polynomial* $\widehat{\mathbf{A}_1} \cdot \widehat{\mathbf{A}_2} \cdot \cdots \cdot \widehat{\mathbf{A}_n}$.

Now fix a field $\mathbb{F}$, and let $\mathbf{A}_k \in \mathbb{F}^{d_{k-1} \times d_k}$ for $k = 1, \ldots, n$ be a collection of matrices over $\mathbb{F}$. Let $\mathbf{R}_k$ be $d_k \times d_k$ matrices of variables for $k \in [n]$, and let $\mathbf{R}_k^{adj}$ be the adjugate matrix of $\mathbf{R}_k$. Let $\mathbf{R}_0 = \mathbf{R}_{n+1} = 1$. Now suppose we set

$$\widehat{\mathbf{A}_k} = \mathbf{R}_{k-1}^{adj} \cdot \mathbf{A}_k \cdot \mathbf{R}_k$$

Theorem 4. *Let $\mathbb{F}, d_k, \mathbf{A}_k, \mathbf{R}_k, \widehat{\mathbf{A}_k}$ be as above. Consider an allowable polynomial p in the $\widehat{\mathbf{A}_k}$, and suppose p, after making the substitution $\widehat{\mathbf{A}_k} = \mathbf{R}_{k-1}^{adj} \cdot \mathbf{A}_k \cdot \mathbf{R}_k$, is identically 0 as a polynomial over the $\mathbf{R}_k$. Then the following is true:*

- *If* $\mathbf{A}_1 \cdot \mathbf{A}_2 \cdot \cdots \cdot \mathbf{A}_n \neq 0$*, then* p *is identically zero as a polynomial over its formal variables, namely the* $\widehat{\mathbf{A}_k}$*.*
- *If* $\mathbf{A}_1 \cdot \mathbf{A}_2 \cdot \cdots \cdot \mathbf{A}_n = 0$ *but*

$$\mathbf{A}_1 \cdot \mathbf{A}_2 \cdot \cdots \cdot \mathbf{A}_{n-1} \neq 0^{1 \times d_n}$$
$$\mathbf{A}_2 \cdot \cdots \cdot \mathbf{A}_{n-1} \cdot \mathbf{A}_n \neq 0^{d_2 \times 1}$$

then p*, as a polynomial over the* $\widehat{\mathbf{A}_k}$*, is a constant multiple of the matrix product polynomial* $\widehat{\mathbf{A}_1} \cdot \widehat{\mathbf{A}_2} \cdot \cdots \cdot \widehat{\mathbf{A}_n}$*.*

Proof. If $n = 1$, there are no $\mathbf{R}_k$ matrices, a single $\mathbf{A}_1$ matrix of dimension 1×1, with entry a. Then $p = p(a) = ca$ for some constant c. As a polynomial over the (non-existent) $\mathbf{R}_i$ matrices, p is just a constant polynomial, so $p = 0$ means $ca = 0$. In the first case above, $a \neq 0$, so $c = 0$, meaning p is identically 0. The second case above is trivially satisfied since the matrix product polynomial is also a constant.

We will assume that $\mathbf{A}_1$ is non-zero in every coordinate. At the end of the proof, we will show this is without loss of generality.

Now we proceed by induction on n. Assume Theorem 4 is proved for $n - 1$. Consider an arbitrary allowable polynomial p. We can write p as

$$p = \sum_{j_1, i_2, j_2, \ldots, j_n, i_{n+1}} \alpha_{j_1, i_2, \ldots, j_{n-1}, i_n} \widehat{A}_{1,1,j_1} \widehat{A}_{2,i_2,j_2} \cdots \widehat{A}_{n-1,i_{n-1},j_{n-1}} \widehat{A}_{n,i_n,1}$$

where $i_{k+1}, j_k \in [d_k]$, and $\widehat{A}_{k,i,j}$ is the (i,j) entry of the matrix $\widehat{\mathbf{A}_k}$. From this point forward, for convenience, we will no longer explicitly refer to the bounds d_k on the i_{k+1}, j_k.

Now we can expand p in terms of the R_1 matrix:

$$\begin{aligned} p &= \sum_{j_1, i_2, j_2, \ldots, j_n, i_{n+1}, m, \ell} \alpha_{j_1, i_2, \ldots, j_{n-1}, i_n} A_{1,1,m} R_{1,m,j_1} R^{adj}_{1,i_2,\ell} (\mathbf{A}_2 \cdot \mathbf{R}_2)_{\ell, j_2} \widehat{A}_{3,i_3,j_3} \cdots \widehat{A}_{n,i_n,1} \\ &= \sum_{j,i,\ell,m} \alpha'_{j,i,\ell} A_{1,1,m} R_{1,m,j} R^{adj}_{1,i,\ell} \end{aligned}$$

where

$$\alpha'_{j,i,\ell} = \sum_{j_2, \ldots, j_n, i_{n+1}} \alpha_{j,i,\ldots,j_{n-1},i_n} (\mathbf{A}_2 \cdot \mathbf{R}_2)_{\ell, j_2} \widehat{A}_{3,i_3,j_3} \cdots \widehat{A}_{n,i_n,1}$$

Recall that

$$R^{adj}_{1,i,\ell} = \sum_{\sigma : \sigma(i) = \ell} \mathsf{sign}(\sigma) \left(\prod_{t \neq i} R_{1,\sigma(t),t} \right)$$

where the sum is over all permutations satisfying $\sigma(i) = \ell$. Thus we can write p as

$$p = \sum_{j,i,\sigma,m} \mathsf{sign}(\sigma) \alpha'_{j,i,\sigma(i)} A_{1,1,m} R_{1,m,j} \left(\prod_{t \neq i} R_{1,\sigma(t),t} \right)$$

Now, since p is identically zero as a polynomial over the $\mathbf{R}_k$ matrices, it must be that for each product $R_{1,m,j}\left(\prod_{t\neq i} R_{1,\sigma(t),t}\right)$, the coefficient of the product (which is a polynomial over the $\mathbf{R}_k : k \geq 2$ matrices) must be identically 0. We now determine the coefficients.

First, we examine the types of products of entries in $\mathbf{R}_1$ that are possible. Products can be thought of as arising from the following process. Choose a permutation σ, which corresponds to selecting d_1 entries of $\mathbf{R}_1$ such that each row and column of $\mathbf{R}_1$ contain exactly one selected entry. Then, for some i, un-select the selected entry from column i and instead select any entry from $\mathbf{R}_1$ (possibly selecting the same entry twice). We observe that the following products are possible:

- $\prod_t R_{1,\sigma(t),t}$ for a permutation σ. This corresponds to re-selecting the un-selected entry from column i. The resulting list of entries determines the permutation σ used to select the original entries (since it is identical to the original list), but allows the column i of the un-selected/re-selected entry to vary. Thus in the summation above, this fixes σ, $j = i$ and $m = \sigma(i)$, but allows i to vary over all values, corresponding to the fact that if we remove any entry and replace it with itself, the result is independent of which entry we removed. Call such products *well-formed*. Well-formed products give the following equation:

$$\sum_i \alpha'_{i,i,\sigma(i)} A_{1,1,\sigma(i)} = 0 \text{ for all } \sigma \tag{1}$$

- $R_{1,m,j} \prod_{t\neq i} R_{1,\sigma(t),t}$ where $j \neq i$ and $m \neq \sigma(i)$. This corresponds to, after un-selecting the entry in column i, selecting a another entry that is in both a different row and a different column. Note that, given final list of selected entries, it is possible to determine the newly selected entry as the unique selected entry that shares both a column with another selected entry and a row with another selected entry. It is also possible to determine the un-selected entry as the only entry that shares no column nor row with another entry. Therefore, the original entry selection is determined as well. Thus, in the summation above, the selected entries fix σ, i, j, and m. In other words, there is no other selection process that gives the same list of entries from $\mathbf{R}_1$. We call such products *malformed type 1*. Malformed type 1 products have the coefficient

$$\alpha'_{j,i,\sigma(i)} A_{1,1,m}$$

  Given any $i, j \neq i, m, \ell \neq m$, pick σ so that $\sigma(i) = \ell$. Since $A_{1,1,m} \neq 0$ for all m, this gives

$$\alpha'_{j,i,\ell} = 0 \text{ for all } i, j \neq i, \ell \tag{2}$$

- $R_{1,m,i} \prod_{t\neq i} R_{1,\sigma(t),t}$ where $m \neq \sigma(i)$. This corresponds to, after un-selecting the entry $R_{1,\sigma(i),i}$, selecting a different entry $R_{1,m,i}$ in the same column. Let i', m', σ' be some other selection process that leads to the same product.

Given the final selection of entries, it is possible to determine $m' = m$ as the only row with two selected entries. It is also possible to determine $\sigma'(i') = \sigma(i)$ as the only row with no selected entries (though i' has not been determined yet). Moreover, i' must be one of the two columns selected in row m, call the other i''. All entries outside of these two rows must have come from the original selection of entries, so this determines $\sigma'(t) = \sigma(t)$ on all inputs outside of i, i''. Notice that if $i = i'$, then σ' agrees with σ on $d_1 - 1$ entries, and since they are both permutations, this sets $\sigma' = \sigma$. In this case, $(i', m', \sigma') = (i, m, \sigma)$.

Otherwise $i' \neq i$, so $i'' = i$, which leaves $\sigma'(i) = \sigma(i') = m$. At this point, σ' is fully determined as $\sigma \circ (i \quad i')$ where $(i \quad i')$ is the transposition swapping i and i'. Therefore, there are two possibilities leading to this product, one corresponding to i and the other corresponding to i'.

We call these products *malformed type 2*. Notice that σ' and σ only differ by a transposition swapping i and i', and so they have opposite parity, meaning the corresponding terms in p have the opposite sign. Given $i, i' \neq i, m, \ell \neq m$, choose σ so that $\sigma(i) = \ell$. This gives us $(\alpha'_{i,i,\ell} - \alpha'_{i',i',\ell}) A_{1,1,m} = 0$. Since $A_{1,1,m} \neq 0$ for all m, we therefore have that $\alpha'_{i,i,\ell} = \alpha'_{i',i',\ell}$ for all i, i'. We can thus choose β_ℓ such that:

$$\alpha'_{i,i,\ell} = \beta_\ell \text{ for all } i, \ell \tag{3}$$

– $R_{1,\sigma(i),j} \prod_{t \neq i} R_{1,\sigma(t),t}$ where $j \neq i$. We call such products *malformed type 3*. The coefficients of these products are linear combinations of the $\alpha'_{i,j,\ell}$ for $i \neq j$, which we already know to be 0. Therefore, these equations are redundant, and we will not need to consider them.

Setting $\sigma(i) = i$ in Eq. 1 and combining with Eq. 3, we have that

$$\sum_\ell \beta_\ell A_{1,1,\ell} = 0 \tag{4}$$

Now we can expand $\alpha'_{j,i,\ell}$ and β_i in Eqs. 2 and 4, obtaining:

$$0 = \alpha'_{i,j,\ell} = \sum_{j_2, i_3, \ldots, j_{n-1}, i_n} \alpha_{j,i,j_2,i_3,\ldots,j_{n-1},i_n} (\mathbf{A}_2 \cdot \mathbf{R}_2)_{\ell,j_2} \widehat{A}_{3,i_3,j_3} \cdots \widehat{A}_{n,i_n,1} \text{ for all } \ell, i, j \neq i \tag{5}$$

$$\begin{aligned} 0 = \sum_\ell \beta_\ell A_{1,1,\ell} &= \sum_{\ell, j_2, i_3, \ldots, j_{n-1}, i_n} \alpha_{i,i,j_2,i_3,\ldots,j_{n-1},i_n} A_{1,1,\ell} (\mathbf{A}_2 \cdot \mathbf{R}_2)_{\ell,j_2} \widehat{A}_{3,i_3,j_3} \cdots \widehat{A}_{n,i_n,1} \\ &= \sum_{j_2, i_3, \ldots, j_{n-1}, i_n} \alpha_{i,i,j_2,i_3,\ldots,j_{n-1},i_n} (\mathbf{A}_1 \cdot \mathbf{A}_2 \cdot \mathbf{R}_2)_{1,j_2} \widehat{A}_{3,i_3,j_3} \cdots \widehat{A}_{n,i_n,1} \text{ for all } i \end{aligned} \tag{6}$$

Now we invoke the inductive step multiple times. Let $\mathbf{A}_{2,\ell}$ be the ℓth row of $\mathbf{A}_2$, and let $\widehat{\mathbf{A}_{2,\ell}} = \mathbf{A}_{2,\ell} \cdot \mathbf{R}_2$. Since $\mathbf{A}_2 \cdot \mathbf{A}_3 \ldots \mathbf{A}_n \neq 0$, there is some ℓ such that $\mathbf{A}_{2,\ell} \cdot \mathbf{A}_3 \ldots \mathbf{A}_n \neq 0$. Then the matrices $\mathbf{A}_{2,\ell}, \mathbf{A}_3, \ldots, \mathbf{A}_n$ satisfy the first set of requirements of Theorem 4 for $n - 1$. Moreover, the right side of Eq. 5

gives an allowable polynomial that is identically zero as a polynomial over the $\mathbf{R}_k, k \geq 2$, and therefore, by induction, it is identically 0 as a polynomial over $\widehat{\mathbf{A}_{2,\ell}}, \widehat{\mathbf{A}_3}, \ldots, \widehat{\mathbf{A}_n}$. This shows us that

$$\alpha_{j,i,j_2,i_3,\ldots,j_{n-1},i_n} = 0 \text{ for all } j \neq i \tag{7}$$

Next, Let $\mathbf{A}'_2 = \mathbf{A}_1 \cdot \mathbf{A}_2$, and let $\widehat{\mathbf{A}'_2} = \mathbf{A}'_2 \cdot \mathbf{R}_2$. There are two cases:

- $\mathbf{A}_1 \cdot \mathbf{A}_2 \cdots \mathbf{A}_n \neq 0$. Then $\mathbf{A}'_2 \cdot \mathbf{A}_3 \cdots \mathbf{A}_n \neq 0$. Therefore, $\mathbf{A}'_2, \mathbf{A}_3, \ldots, \mathbf{A}_n$ satisfy the first set of requirements in Theorem 4. Moreover, for each i, Eq. 6 gives an allowable polynomial that is identically zero as a polynomial over the $\mathbf{R}_k, k \geq 2$. Therefore, by induction, the polymomial is identically zero as a polynomial over $\widehat{\mathbf{A}'_2}, \widehat{\mathbf{A}_3}, \ldots, \widehat{\mathbf{A}_n}$. This means

  $$\alpha_{i,i,j_2,i_3,\ldots,j_{n-1},i_n} = 0 \text{ for all } i$$

  Combining with Eq. 7, we have that all the α values are 0. Therefore p is identically zero as a polynomial over the $\widehat{\mathbf{A}_1}, \widehat{\mathbf{A}_2}, \ldots, \widehat{\mathbf{A}_n}$.
- $\mathbf{A}_1 \cdot \mathbf{A}_2 \cdots \mathbf{A}_n = 0$. Then $\mathbf{A}'_2 \cdot \mathbf{A}_3 \cdots \mathbf{A}_n = 0$. However, $\mathbf{A}'_2 \cdot \mathbf{A}_3 \cdots \mathbf{A}_{n-1} = \mathbf{A}_1 \cdot \mathbf{A}_2 \cdots \mathbf{A}_{n-1} \neq 0$ and $\mathbf{A}_3 \ldots \mathbf{A}_4 \cdots \mathbf{A}_n \neq 0$ (since otherwise $\mathbf{A}_2 \cdots \mathbf{A}_3 \cdots \mathbf{A}_n = 0$, contradicting the assumptions of Theorem 4). Therefore, $\mathbf{A}'_2, \mathbf{A}_3, \ldots, \mathbf{A}_n$ satisfy the second set of requirements in Theorem 4. By induction, for each i, the polynomial in Eq. 6 must therefore be a multiple $\gamma_i \widehat{\mathbf{A}'_2} \cdot \widehat{\mathbf{A}_3} \cdots \widehat{\mathbf{A}_n}$ of the matrix product polynomial. This is equivalent to

  $$\begin{aligned} \alpha_{i,i,j_2,i_3,\ldots,j_{n-1},i_n} &= 0 \text{ if } j_k \neq i_{k+1} \text{ for any } k \\ \alpha_{i,i,i_3,i_3,\ldots,i_n,i_n} &= \gamma_i \end{aligned}$$

  This means we can write

  $$\begin{aligned} &\alpha'_{j,i,\ell} = 0 \text{ for all } j \neq i \text{ (by Eq. 7 and the definition of } \alpha'_{i,j,\ell}) \\ &\alpha'_{i,i,\ell} = \gamma_i \sum_{i_3,\ldots,i_n} (\mathbf{A}_2 \cdot \mathbf{R}_2)_{\ell,i_3} \widehat{A}_{3,i_3,i_4} \ldots \widehat{A}_{n,i_n,1} = \gamma_i \left(\mathbf{A}_2 \cdot \mathbf{A}_3 \cdots \mathbf{A}_n\right)_{\ell,1} \end{aligned}$$

  Since $\alpha'_{i,i,\ell} = \beta_\ell$ for all i and the product $\mathbf{A}_2 \cdot \mathbf{A}_3 \cdots \mathbf{A}_n$ is non-zero, we have that $\gamma_i = \gamma$ is the same for all i. Therefore,

  $$\alpha_{i,i,i_3,i_3,\ldots,i_n,i_n} = \gamma \text{ for all } i, i_3, \ldots, i_n$$

  meaning p is a multiple of the matrix product polynomial, as desired.

It remains to show the case where $\mathbf{A}_1$ has zero entries. Since $\mathbf{A}$ is non-zero (as a consequence of our assumptions), and $\mathbf{A}$ is a single row vector, it is straightforward to build an invertible matrix $\mathbf{B}$ such that $\mathbf{A}'_1 = \mathbf{A}_1 \cdot \mathbf{B}$ is non-zero in every coordinate.

Let $\mathbf{A}'_2 = \mathbf{B}^{-1}\mathbf{A}_2$. Let $\mathbf{R}'_1 = \mathbf{B}^{-1} \cdot \mathbf{R}_1$, $\widehat{\mathbf{A}'_1} = \mathbf{A}'_1 \cdot \mathbf{R}'_1 = \widehat{\mathbf{A}_1}$, and $\widehat{\mathbf{A}'_2} = (\mathbf{R}'_1)^{adj} \cdot \mathbf{A}'_2 \cdot \mathbf{R}_2 = \widehat{\mathbf{A}_2}$. Now $\mathbf{A}'_1, \mathbf{A}'_2, \mathbf{A}_3, \ldots, \mathbf{A}_n$ satisfy the same conditions

of Theorem 4 as the original $\mathbf{A}_k$. Moreover, p is still allowable as a polynomial over $\widehat{\mathbf{A}'_1}, \widehat{\mathbf{A}'_2}, \widehat{\mathbf{A}_3}, \ldots \widehat{\mathbf{A}_n}$. Moreover, we can relate p as a polynomial over $\mathbf{R}_k$ to p as a polynomial over $\mathbf{R}'_1, \mathbf{R}_2, \ldots, \mathbf{R}_{n-1}$ by a linear transformation on the $\mathbf{R}_1$ variables. Therefore, p is identically zero as a polynomial over the $\mathbf{R}_k$ if and only if it is identically zero as a polynomial over $\mathbf{R}'_1, \mathbf{R}_2, \ldots, \mathbf{R}_n$. Thus we can invoke Theorem 4 on $\mathbf{A}'_1, \mathbf{A}'_2, \ldots, \mathbf{A}_n$ using the same polynomial p, and arrive at the desired conclusion. This completes the proof.

5 Sketch of VBB Security Proof

We now explain how to use Theorem 4 to prove the VBB security of our obfuscator. Due to space constraints, the complete proof is deferred to the full version of this paper. In this sketch, we pay special attention to the steps in our proof that deviate from previous works [1,6]. We also state a definition and lemma that will be used in Sect. 6 to prove that encodings of zero cannot be created when the function being obfuscated is evasive.

The adversary is given an obfuscation of a branching program BP, which consists of a list of handles corresponding to elements in the graded encoding. The adversary can operate on these handles using the graded encoding interface, which allows performing algebraic operations and zero testing. Our goal is to build a simulator that has oracle access only to the output of BP, and is yet able to simulate all of the handles and interfaces seen by the adversary. Formally, we prove the following theorem.

Theorem 5. *If BP^S outputs non-shortcutting branching programs, then for any PPT adversary $\mathcal{A}$, there is a PPT simulator* Sim *such that*

$$\left|\Pr[\mathcal{A}^{\mathcal{M}}(\mathcal{O}^{\mathcal{M}}(BP^S)) = 1] - \Pr_{BP \leftarrow BP^S}[\mathsf{Sim}^{BP}(\ell, d_0, \ldots, d_\ell, \mathsf{inp}_0, \mathsf{inp}_1) = 1]\right| < \mathsf{negl}.$$

The simulator will choose random handles for all of the encodings in the obfuscation, leaving the actual entries of the $\mathbf{D}_{i,b_0,b_1}$ as formal variables[5]. Simulating the algebraic operations is straightforward; the bulk of the security analysis goes in to answering zero-test queries. Any handle the adversary queries the zero test oracle on corresponds to some polynomial p on the variables $\mathbf{D}_{i,b_0,b_1}$, which the adversary can determine by inspecting the queries made by the adversary so far.

The simulator's goal is to decide if p evaluates to zero, when the formal variables in the $\mathbf{D}_{i,b_0,b_1}$ are set to the values in the randomized matrix branching program BP'. However, the simulator does not know BP', and must instead determine if p gives zero knowing only the outputs of BP.

The analysis of [1,6] first simplifies the problem of determining if p evaluates to zero, using Lemma 3 below.

[5] The simulator does not know the branching program, and so it has no way of actually sampling the $\mathbf{D}_{i,b_0,b_1}$.

Definition 11. *A* single-input element *for an input x is a polynomial p_x whose variables are the $\mathbf{C}_{i,x_{\mathsf{inp}_0(i)},x_{\mathsf{inp}_1(i)}}$ matrices, and p_x is allowable in the sense of Definition 10: each monomial in the expansion of p_x contains exactly one variable from each of the $\mathbf{C}_{i,x_{\mathsf{inp}_0(i)},x_{\mathsf{inp}_1(i)}}$ matrices.*

Lemma 3 (Adapted from [1,6]). *Any polynomial p over the obfuscation $\mathcal{O}^{\mathcal{M}}(BP^S)$ can be efficiently decomposed into a sum $p = \sum_{x\in D} \alpha_x p_x$, where $\alpha_x = \prod_{i\in[\ell]} \alpha_{i,x_{\mathsf{inp}_0(i)},x_{\mathsf{inp}_1(i)}}$, each p_x is a single-input element for input x, and $|D|$ is polynomial in the circuit size of p.*

Due to the independence of the α_x variables, it can be shown that p evaluates to zero iff each of the polynomials p_x do. Thus Lemma 3, along with some extra analysis of our own to handle multi-bit outputs, reduces the general problem to the following simpler problem. There is an unknown sequence of matrices $\mathbf{A}_i \in \mathbb{Z}_q^{d_{i-1}\times d_i}$ for $i \in [\ell]$, where $d_0 = d_\ell = 1$ (the shapes of the $\mathbf{A}_i$ ensure that the product $\prod_{i\in[\ell]} \mathbf{A}_i$ is valid and results in a scalar). We are also given an allowable polynomial p' on matrices of random variables $\widehat{\mathbf{A}_i}$. Our goal is to determine, if the $\widehat{\mathbf{A}_i}$ are set to the Kilian-randomized matrices $\widehat{\mathbf{A}_i} = \mathbf{R}_{i-1}\cdot\mathbf{A}\cdot\mathbf{R}_i^{adj}$, whether or not p' evaluates to zero. We note that by applying the Schwartz-Zippel lemma, it suffices to decide if p' is *identically* zero, when considered a polynomial over the formal variables $\mathbf{R}_i$.

It is not hard to see that this simpler problem is impossible in general: p' could be the polynomial computing the iterated matrix product $\prod_{k\in[\ell]} \widehat{\mathbf{A}_i}$, which is equal to $\prod_{i\in[\ell]} \mathbf{A}_i$. Therefore, to decide if p' is identically zero in this case, we at a minimum need to know if $\prod_{i\in[\ell]} \mathbf{A}_i$ evaluates to 0.

The analysis shows that the $\mathbf{A}_i$ are actually equal to $\mathbf{B}_{i,x_{\mathsf{inp}_0(i)},x_{\mathsf{inp}_1(i)}}$ for some (known) input x, where $\mathbf{B}_{i,b_0,b_1}$ are the matrices in the branching program BP. Therefore, we can determine if $\prod_{i\in[\ell]} \mathbf{A}_i = 0$ by querying the BP oracle on x. In the case where p' is the iterated matrix product, this allows us to determine if p' is identically 0. What about other, more general, polynomials p'?

In previous works, $\mathbf{A}_1$ and $\mathbf{A}_\ell$ are bookend vectors, and the $\mathbf{A}_i$ for $k \in [2, \ell-1]$ are square invertible matrices. In this setting, Kilian's statistical simulation theorem allows us to sample from the distribution of $\widehat{\mathbf{A}_i}$ knowing only the product of the $\mathbf{A}_i$, but not the individual values. Then we can apply p' to the sample, and the Schwartz-Zippel lemma shows that p' will evaluate to zero, with high probability, if and only if it is identically zero. This allows deciding if p' is identically zero.

In our case, we cannot sample from the correct distribution of $\widehat{\mathbf{A}_i}$. Instead, we observe that our branching program is non-shortcutting, which means the $\mathbf{A}_i$ and p' satisfy the requirements of Theorem 4. Theorem 4 implies something remarkably strong: if p' is not (a multiple of) the iterated matrix product, it *cannot possibly* be identically zero as a polynomial over the formal variables $\mathbf{R}_k$. Thus, we first decide if p' is a multiple of the iterated matrix product, which is possible using the Schwartz-Zippel lemma. If p' is a multiple, then we know it is identically zero if and only if the product $\prod_{i\in[\ell]} \mathbf{A}_i$ is zero, and we know whether this product is zero by using our BP oracle.

6 Obfuscating Evasive Functions with No Zero Encodings

In this section we show that when the obfuscator of Sect. 3 is applied to an *evasive* function, any poly-time adversary will have only negligible probability in constructing an encoding of 0.

Definition 12. *We say that an adversary $\mathcal{A}$* constructs an encoding of 0 *if it ever receives a handle h from $\mathcal{M}$ such that (a) h maps to an encoding of 0 in $\mathcal{M}$'s table, and (b) the polynomial that produced the encoding is not identically zero as a polynomial over its formal variables.*

Theorem 6. *Let $\mathcal{O}$ be the obfuscator from Sect. 3, and let BP^S sample an evasive function family. Then for any PPT adversary $\mathcal{A}$:*

$$\Pr\left[\mathcal{A}^{\mathcal{M}}(\mathcal{O}^{\mathcal{M}}(BP^S)) \text{ constructs an encoding of } 0\right] < \mathsf{negl}(\ell).$$

One can never prevent an adversary from constructing a trivial encoding of 0 by computing $e - e$ for some encoding e that it has. (More generally, any identically zero polynomial will produce a trivial encoding of 0.) However in all candidate constructions of graded encoding schemes, such an operation always produces *the integer* 0, which contains no information. Indeed, it seems unlikely that a plausible candidate would not have this property.

To prove Theorem 6, we first show that any element that is not at the top level $\mathbb{U}$ can be "completed" to the top level by multiplying with other basic elements output by the obfuscator. This is a consequence of our use of strong straddling sets.

Definition 13. *For $i \in [\ell]$ and $b \in \{0,1\}$, an element encoded at level S_{j,b_0,b_1}* implies $x_i = b$ *if either $\mathsf{inp}_0(j) = i$ and $b_0 = b$ or $\mathsf{inp}_1(j) = i$ and $b_1 = b$.*

Lemma 4. *Let $R := \{[\mathbf{D}_{i,b_0,b_1}]_{S_{i,b_0,b_1}}\}$ be the basic elements output by the obfuscator $\mathcal{O}$, and let $[r]_S$ be any valid element created by a polynomial p over R.*

Then there exists a set of elements $R' \subseteq R$ such that $[r]_S \times \prod_{z \in R'} z$ is a valid element at level $\mathbb{U}$, and further R' can be efficiently found.

Proof. We say that p *touches* layer $j \in [n]$ if any leaf of p is a basic element from layer j (cf. [30, Definition 4.2]). S uniquely determines the layers touched by p and vice versa (though not necessarily the specific matrices touched in each layer); in particular, p touches every layer iff $S = \mathbb{U}$. Thus we construct R' to contain one basic element from each layer that is not touched by p. If $S = \mathbb{U}$ then the lemma holds trivially with $R' := \emptyset$, so assume $S \neq \mathbb{U}$ and let $J \subseteq [n]$ be the set of layers not touched by p. Let $I := \{\mathsf{inp}_0(j), \mathsf{inp}_1(j) \mid j \in J\} \subseteq [\ell]$ be the set of all indices that are read in some untouched layer.

We claim that there is a sequence $(b_i)_{i \in I} \in \{0,1\}^{|I|}$ such that for every $i \in I$, p's leaves do not contain any basic element that implies $x_i = 1 - b_i$. Fix any $i \in I$. Recall that $\mathbb{U}_i \subset \mathbb{U}$ is the universe set for index i, and note that we must have $\mathbb{U}_i \not\subseteq S$ because some layer that reads index i is untouched. If $\mathbb{U}_i \cap S = \emptyset$,

then p's leaves do not contain a basic element that implies $x_i = 0$ nor one that implies $x_i = 1$; in this case we can take $b_i = 0$. If instead $\mathbb{U}_i \cap S \notin \{\emptyset, \mathbb{U}_i\}$, then by Lemma 1 there is a unique $b_i \in \{0,1\}$ for which there exists $J' \subset [n]$ such that

$$\mathbb{U}_i \cap S = \bigcup_{j' \in J'} S^i_{j',b_i}.$$

(Recall that each S^i_{j',b_i} comes from the strong straddling set system over $\mathbb{U}_i$.) Thus p's leaves do not contain any basic element that implies $x_i = 1 - b_i$.

Finally let R' contain, for each $j \in J$, an arbitrary entry from the $(b_{\mathsf{inp}_0(j)}, b_{\mathsf{inp}_1(j)})$th matrix in layer j. Formally, $R' := \{\mathbf{D}_{j,b_{\mathsf{inp}_0(j)},b_{\mathsf{inp}_1(j)}}[0,0] \mid j \in J\}$ which can be efficiently computed given e. Then $[r]_S \times \prod_{z \in R'} z$ is valid by construction, and it is at level $\mathbb{U}$ because it touches every layer.

We now prove the main theorem of this section. The proof uses the simulator Sim of Theorem 5 in a non-black-box way, and specifically relies on properties of the decomposition $p = \sum_x \alpha_x p_x$ given by Lemma 3.

Proof *(Proof of Theorem 6).* For any PPT adversary $\mathcal{A}$, denote

$$\mathcal{P}'(\mathcal{A}) := \Pr\left[\mathcal{A}^{\mathcal{M}}(\mathcal{O}^{\mathcal{M}}(BP^S)) \text{ constructs a level-}\mathbb{U} \text{ encoding of } 0\right].$$

We first show that if $\mathcal{P}'(\mathcal{A})$ is a noticeable function of ℓ for some PPT $\mathcal{A}$, then BP^S cannot be evasive, in contradiction to our assumption. Next we use Lemma 4 to remove the assumption that $\mathcal{A}$'s encoding of 0 is at level $\mathbb{U}$.

Let $f \leftarrow BP^S$ denote the function being obfuscated. Let $\mathcal{A}$ be any PPT, and let Sim denote the corresponding simulator given by Theorem 5. We construct a new adversary $\mathcal{B}$, with oracle access to f, that finds an input x such that $f(x) = 0$.

$\underline{\mathcal{B}^f\left(1^\ell\right)}$:

1. Run Sim^f, which itself is running $\mathcal{A}$, up until the point where $\mathcal{A}$ constructs a level-$\mathbb{U}$ encoding.
2. Decompose $p = \sum_{x \in D} \alpha_x p_x$ as in Lemma 3. Check if $f(x) = 0$ for any $x \in D$. If so, stop and output x; otherwise, continue running Sim until $\mathcal{A}$'s next level-$\mathbb{U}$ encoding, and repeat.
3. If Sim halts, then output a random $x \in \{0,1\}^\ell$.

Note that $\mathcal{B}$'s simulation of $\mathcal{A}$'s view is correct up to statistical distance $\mathsf{negl}(\ell)$, because Sim's is. The proof of Theorem 5 establishes that for any level-$\mathbb{U}$ p constructed by $\mathcal{A}$,

$$\Pr[p \text{ is an encoding of 0 but some } p_x \text{ is not}] < \mathsf{negl}(\ell).$$

Further, Theorem 4 establishes that if p_x is not identically zero (and some p_x must not be since p is not), then p_x is a multiple of the honest matrix product polynomial corresponding to input x. Thus p_x is an encoding of 0 iff $f(x) = 0$, and we have established $\forall$ PPT $\mathcal{A}$ $\exists$ PPT $\mathcal{B}$:

$$\Pr\left[f\left(\mathcal{B}^f(1^\ell)\right) = 0\right] \geq \mathcal{P}'(\mathcal{A}) - \mathsf{negl}(\ell). \tag{8}$$

Finally, let

$$\mathcal{P}(\mathcal{A}) := \Pr\left[\mathcal{A}^{\mathcal{M}}(\mathcal{O}^{\mathcal{M}}(BP^S)) \text{ constructs an encoding of } 0\right]$$

be the probability that we want to bound. We claim that $\forall$ PPT $\mathcal{A}$ $\exists$ PPT $\mathcal{A}'$: $\mathcal{P}'(\mathcal{A}') \geq \mathcal{P}(\mathcal{A})$. Namely $\mathcal{A}'$ runs $\mathcal{A}$, and for every encoding $[r]_S$ with $S \neq \mathbb{U}$ created by $\mathcal{A}$, $\mathcal{A}'$ also creates the level-$\mathbb{U}$ encoding $[r']_{\mathbb{U}} := [r]_S \times \prod_{z \in R'} z$ guaranteed by Lemma 4. Note that if $[r]_S$ encodes 0 then $[r']_{\mathbb{U}}$ must encode 0 as well, so we have $\mathcal{P}'(\mathcal{A}') \geq \mathcal{P}(\mathcal{A})$. Combining this with (8), we complete the proof: if $\exists$ PPT $\mathcal{A}$ such that $\mathcal{P}(\mathcal{A})$ is a noticeable function of ℓ, then BP^S does not sample an evasive function family.

In the full version of this paper, we show that, via the bootstrapping technique of [12,20], an obfuscator for log-depth evasive circuits implies an obfuscator for all poly-size evasive circuits.

References

1. Ananth, P.V., Gupta, D., Ishai, Y., Sahai, A.: Optimizing obfuscation: avoiding Barrington's theorem. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, pp. 646–658 (2014)
2. Apon, D., Huang, Y., Katz, J., Malozemoff, A.J.: Implementing cryptographic program obfuscation. Cryptology ePrint Archive, Report 2014/779 (2014). http://eprint.iacr.org/
3. Applebaum, B.: Bootstrapping obfuscators via fast pseudorandom functions. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014, Part II. LNCS, vol. 8874, pp. 162–172. Springer, Heidelberg (2014)
4. Applebaum, B., Brakerski, Z.: Obfuscating circuits via composite-order graded encoding. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015, Part II. LNCS, vol. 9015, pp. 528–556. Springer, Heidelberg (2015)
5. Barak, B., Bitansky, N., Canetti, R., Kalai, Y.T., Paneth, O., Sahai, A.: Obfuscation for evasive functions. In: Lindell, Y. (ed.) TCC 2014. LNCS, vol. 8349, pp. 26–51. Springer, Heidelberg (2014)
6. Barak, B., Garg, S., Kalai, Y.T., Paneth, O., Sahai, A.: Protecting obfuscation against algebraic attacks. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 221–238. Springer, Heidelberg (2014)
7. Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S.P., Yang, K.: On the (im)possibility of obfuscating programs. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 1–18. Springer, Heidelberg (2001)
8. Boneh, D., Lewi, K., Raykova, M., Sahai, A., Zhandry, M., Zimmerman, J.: Semantically secure order-revealing encryption: multi-input functional encryption without obfuscation. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 563–594. Springer, Heidelberg (2015)
9. Boneh, D., Wu, D.J., Zimmerman, J.: Immunizing multilinear maps against zeroizing attacks. Cryptology ePrint Archive, Report 2014/930 (2014). http://eprint.iacr.org/
10. Boneh, D., Zhandry, M.: Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part I. LNCS, vol. 8616, pp. 480–499. Springer, Heidelberg (2014)

11. Brakerski, Z., Gentry, C., Halevi, S., Lepoint, T., Sahai, A., Tibouchi, M.: Cryptanalysis of the quadratic zero-testing of GGH. Cryptology ePrint Archive, Report 2015/845 (2015). http://eprint.iacr.org/
12. Brakerski, Z., Rothblum, G.N.: Virtual black-box obfuscation for all circuits via generic graded encoding. In: Lindell, Y. (ed.) TCC 2014. LNCS, vol. 8349, pp. 1–25. Springer, Heidelberg (2014)
13. Cheon, J.H., Han, K., Lee, C., Ryu, H., Stehlé, D.: Cryptanalysis of the multilinear map over the integers. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 3–12. Springer, Heidelberg (2015)
14. Cheon, J.H., Lee, C., Ryu, H.: Cryptanalysis of the new clt multilinear maps. Cryptology ePrint Archive, Report 2015/934 (2015). http://eprint.iacr.org/
15. Coron, J.-S., et al.: Zeroizing without low-level zeroes: new MMAP attacks and their limitations. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9216, pp. 247–266. Springer, Heidelberg (2015)
16. Coron, J.-S., Lepoint, T., Tibouchi, M.: Practical multilinear maps over the integers. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 476–493. Springer, Heidelberg (2013)
17. Coron, J.-S., Lepoint, T., Tibouchi, M.: New multilinear maps over the integers. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9216, pp. 267–286. Springer, Heidelberg (2015)
18. Garg, S., Gentry, C., Halevi, S.: Candidate multilinear maps from ideal lattices. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 1–17. Springer, Heidelberg (2013)
19. Garg, S., Gentry, C., Halevi, S., Raykova, M.: Two-round secure MPC from indistinguishability obfuscation. In: Lindell, Y. (ed.) TCC 2014. LNCS, vol. 8349, pp. 74–94. Springer, Heidelberg (2014)
20. Garg, S., Gentry, C., Halevi, S., Raykova, M., Sahai, A., Waters, B.: Candidate indistinguishability obfuscation and functional encryption for all circuits. In: FOCS, pp. 40–49 (2013)
21. Garg, S., Gentry, C., Sahai, A., Waters, B.: Witness encryption and its applications. In: STOC (2013)
22. Gentry, C., Gorbunov, S., Halevi, S.: Graph-induced multilinear maps from lattices. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015, Part II. LNCS, vol. 9015, pp. 498–527. Springer, Heidelberg (2015)
23. Gentry, C., Lewko, A., Waters, B.: Witness encryption from instance independent assumptions. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part I. LNCS, vol. 8616, pp. 426–443. Springer, Heidelberg (2014)
24. Gentry, C., Lewko, A.B., Sahai, A., Waters, B.: Indistinguishability obfuscation from the multilinear subgroup elimination assumption. In: IEEE Symposium on Foundations of Computer Science FOCS, pp. 151–170 (2015)
25. Goldwasser, S., et al.: Multi-input functional encryption. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 578–602. Springer, Heidelberg (2014)
26. Goyal, V., Ishai, Y., Sahai, A., Venkatesan, R., Wadia, A.: Founding cryptography on tamper-proof hardware tokens. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978, pp. 308–326. Springer, Heidelberg (2010)
27. Halevi, S.: Graded encoding, variations on a scheme. IACR Cryptology ePrint Archive 2015, 866 (2015)
28. Hu, Y., Jia, H.: Cryptanalysis of GGH map. IACR Cryptology ePrint Archive 2015, 301 (2015)

29. Kilian, J.: Founding cryptography on oblivious transfer. In: STOC, pp. 20–31 (1988)
30. Miles, E., Sahai, A., Weiss, M.: Protecting obfuscation against arithmetic attacks. IACR Cryptology ePrint Archive 2014, 878 (2014)
31. Miles, E., Sahai, A., Zhandry, M.: Annihilation attacks for multilinear maps: Cryptanalysis of indistinguishability obfuscation over ggh13. Cryptology ePrint Archive, Report 2016/147 (2016). http://eprint.iacr.org/
32. Minaud, B., Fouque, P.A.: Cryptanalysis of the new multilinear map over the integers. Cryptology ePrint Archive, Report 2015/941 (2015). http://eprint.iacr.org/
33. Pass, R., Seth, K., Telang, S.: Indistinguishability obfuscation from semantically-secure multilinear encodings. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part I. LNCS, vol. 8616, pp. 500–517. Springer, Heidelberg (2014)
34. Sahai, A., Waters, B.: How to use indistinguishability obfuscation: deniable encryption, and more. In: Symposium on Theory of Computing (STOC), pp. 475–484 (2014)
35. Sahai, A., Zhandry, M.: Obfuscating low-rank matrix branching programs. IACR Cryptology ePrint Archive 2014, 773 (2014). http://eprint.iacr.org/2014/773
36. Zhandry, M.: How to avoid obfuscation using witness PRFs. In: Kushilevitz, E., Malkin, T. (eds.) TCC 2016-A. LNCS, vol. 9563, pp. 421–448. Springer, Heidelberg (2016). doi:10.1007/978-3-662-49099-0_16
37. Zimmerman, J.: How to obfuscate programs directly. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 439–467. Springer, Heidelberg (2015)