Post-Zeroizing Obfuscation: The case of Evasive Circuits*

Saikrishna Badrinarayanan
UCLA
saikrishna@cs.ucla.edu

Eric Miles
UCLA
enmiles@cs.ucla.edu

Amit Sahai†
UCLA
sahai@cs.ucla.edu

Mark Zhandry
Stanford University
mzhandry@stanford.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 present a construction for the special case of evasive functions. We show that our obfuscator survives *all known attacks* on the underlying multilinear maps, by proving that no 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 are now known to be false.

To prove security, we introduce a new technique for analyzing polynomials over multilinear map encodings. 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.

*This paper subsumes a previous work of Sahai and Zhandry [SZ14].

†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.

1 Introduction

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. Then 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 [GGH+13b]. The [GGH+13b] construction, and all subsequent works constructing obfuscators [BR14, BGK+14, PST14a, GLSW14, AGIS14, Zim14, MSW14], are built on top of another cryptographic primitive called a graded encoding scheme, also known as an approximate multilinear map. Candidate graded encoding schemes were given previously by Garg, Gentry, and Halevi [GGH13a] and Coron, Lepoint, and Tibouchi [CLT13]. Some of the obfuscation works prove security in an idealized "generic multilinear model" that seeks to capture the algebraic restrictions imposed by the graded encoding scheme [BR14, BGK+14, AGIS14, Zim14, MSW14], while others prove security in the plain model starting from specific assumptions on the graded encoding scheme [BBC+14, PST14a, GLSW14].

In a graded encoding scheme, plaintext elements are encoded at various levels, and there is a top level at which one can test whether an element encodes 0. It was known previously that low-level encodings of 0 allowed for "zeroizing" attacks on [GGH13a], but recently, Cheon, Han, Lee, Ryu, and Stehlé [CHL+14] demonstrated a new such zeroizing attack on the [CLT13] scheme, again relying on low-level encodings of 0. Notably, none of the proposed obfuscation schemes required low-level encodings of 0 to be given to the adversary, and so it seemed plausible that the security arguments that we had for obfuscation are still persuasive.

However, subsequent concurrent works by Gentry, Halevi, Maji, and Sahai [GHMS14] and Boneh, Wu, and Zimmerman [BWZ14] (see also further follow-up work [CLT14]) showed for the first time that zeroizing attacks are possible even when no low-level encodings of zero are made available to the adversary, just as long as *top-level* encodings of zero can be created. These attacks are particularly devastating in the case of [CLT13], where they lead to a complete break and recover all secret parameters. The attacks all obey the algebraic restrictions of the graded encoding scheme, and in general they have highlighted significant gaps in our intuition about the source of security in graded encoding scheme candidates.

The current state of affairs is worrisome. It is true that some of the above obfuscation schemes [BR14, BGK+14, AGIS14, Zim14, MSW14] are not known to be broken by the new attacks. However, all previous *proofs of security* for obfuscation are no longer persuasive. This is because the new zeroizing attacks show that there are natural and devastating algebraic attacks that are not captured by the generic multilinear model, and therefore a proof in the generic multilinear model fails to rule out real attacks. Similarly, all known reduction-based security arguments for obfuscation [BBC+14, PST14a, GLSW14] reduce to assumptions that are known to be false due to the new zeroizing attacks.

Post-zeroizing obfuscation. The new zeroizing attacks give rise to a pressing question: can there be any persuasive argument of security for obfuscation schemes? To this end, the work of [GHMS14] proposes a new kind of post-zeroizing generic model, that seeks to capture the power that zeroizing attacks offer to an adversary. In this work, we take an even more conservative approach: All known efficient attacks on multilinear map candidates make use of information that is leaked when a *top-level* encoding of 0 is created by the adversary. Thus, we consider a generic model where the adversary "wins" whenever it manages to create an encoding of 0 at *any* level. Thus, a security proof in our generic model would rule out not only any current zeroizing attacks, but even future extensions of zeroizing attacks that exploit information leaked by encoded 0s.

Armed with this generic model, we give the first candidate obfuscator for a natural class of functions that *provably does not allow any encodings of 0 to be constructed*, and we initiate generally the study of this type of security. In this work we show how to obfuscate *evasive* functions [BBC+14], 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 would have 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 [BBC+14] 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 [BBC+14]. However, the positive results in [BBC+14] were based on assumptions over approximate multilinear maps that are now known to be false [CHL+14], and furthermore the positive results in [BBC+14] did not consider completely arbitrary distributions of evasive circuits, as we do here.

Our main theorem is the following.

Theorem 1.1 (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 $\mathcal{A}$,*

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

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

Crucially, this theorem allows us to prove the security of our construction in a generic model which, for the first time, captures *all* known attacks on graded encoding schemes. In previous works that prove security in a generic model, the graded encoding scheme's zero-test function is modeled as a Boolean function (i.e. one that returns a yes/no answer). In candidate constructions however [GGH13a, CLT13], a successful zero-test actually returns an algebraic element in the ring of encodings, and this fact is crucially exploited in the attacks. By contrast, we consider *any* encoding of 0 to be a complete break, thereby capturing these "zeroizing" attacks.

Avoiding encodings of 0. Because we focus on obfuscating evasive functions, we can hope to at least avoid the "easy" encodings of 0 corresponding to an honest evaluation that outputs 0. However, we stress that previous obfuscators do not prevent encodings of 0 *even when the function being obfuscated is evasive.* The obfuscation scheme of Brakerski and Rothblum [BR14] gives a concrete example of this: the obfuscator operates on matrix branching programs constructed using Barrington's theorem [Bar86], in which the product matrix corresponding to any honest function evaluation is either the identity or is another fixed permutation matrix. In either case, the product matrix contains some 0 entries which the adversary then obtains as encodings of 0. Beyond just the [BR14] construction however, no previous obfuscator precluded the adversary from constructing top-level encodings of 0 that may be unrelated to any honest function evaluation.

We introduce a new "core obfuscator" that expands the class of matrix branching programs that can be directly obfuscated. In addition to being a key component of the proof of Theorem 1.1, this obfuscator also gives efficiency improvements. We prove a key technical theorem for this obfuscator (Theorem 4.2), which was not present in any previous work and is likely of independent interest. This theorem shows that any polynomial p over the obfuscation $\mathcal{O}(f)$ can be efficiently mapped to a poly-size set of inputs $X \subset \{0,1\}^n$ such that p evaluates to an encoding of 0 if and only if some $x \in X$ satisfies $f(x) = 0$. Thus, provided that f is evasive, no PPT adversary can create an encoding of 0. For context, previous works 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 condition that we require. Furthermore, our theorem shows for the first time how to obfuscate matrix branching programs that are represented with low-rank matrices, or where the matrices are non-square. This leads to efficiency improvements even beyond previous obfuscators that lack a post-zeroizing proof of security, as detailed in Appendix B. Our analysis also extends to other settings besides obfuscation: for example, Boneh et al. [BLR+14] rely on our analysis to obtain near-practical order-revealing encryption.

We also show that the "bootstrapping" theorem of [GGH+13b] extends to the setting of evasive functions. This theorem transforms a core obfuscator for a "small" class of functions (e.g. poly-size branching programs) into an obfuscator for *all* efficient functions. We observe that this theorem only uses the core obfuscator on evasive functions, and we show that the proof goes through only assuming its security on such functions. In particular, we show that Theorem 1.1 applies to all evasive functions and not only those on which the core obfuscator operates. Interestingly, the bootstrapping technique of Applebaum [App13] cannot be used for our purposes, because it inherently produces encodings of 0 regardless of the function being obfuscated.

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 poly-size set of inputs $X \subset \{0,1\}^n$ such that p evaluates to an encoding of 0 if and only if some $x \in X$ satisfies $f(x) = 0$.

One ingredient in our paper is the notion of strong straddling sets from the recent paper of [MSW14], as this tool allows us to eliminate the possibility of any encodings of 0 below the top level. Thus, the only obstacle that remains is to prove the theorem above for *top-level* encodings of 0.

The technical barrier – Kilian's statistical simulation. Before we proceed to provide intuition about our proof, let us consider the technical roots of how this theorem was avoided in previous papers on obfuscating matrix branching programs. In every paper constructing secure obfuscation for matrix branching programs so far [GGH+13b, BR14, BGK+14, PST14a, GLSW14, AGIS14, MSW14] 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 [Kil88]. 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, note that 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, as pointed out earlier in the case of [BR14], this in fact can actually happen in some obfuscators from the literature that apply Kilian's theorem.

If we are to obtain a method for obfuscation that avoids encodings of 0s altogether for evasive functions, then we would need to avoid Kilian's theorem entirely, deviating from all previous analyses of obfuscation.

Our Approach. Our key technical challenge is to replace the use of Kilian's theorem [Kil88] from previous generic model proofs for obfuscation (e.g. [BGK+14, AGIS14]). Nevertheless, even though we will not rely on Kilian's simulation theorem, we will use a matrix randomization scheme that is essentially[1] identical to the one used when applying Kilian's randomization.

To obtain our result, we must directly analyze what kinds of polynomials an adversary can generate using multilinear operations. Before continuing, we remark that our analysis at this stage will not be efficient. Nevertheless, this analysis will allow us to obtain an efficient simulator in our generic model.

Roughly speaking, we can model the multilinear setting as follows: There is a universe set $[\ell]$. For every subset $S \subset [\ell]$, 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 $S \subset [\ell]$.
- Mult: $G_S \times G_T \to G_{S \cup T}$, for every pair $S, T \subset [\ell] : S \cap T = \emptyset$.
- ZeroTest: $G_{[\ell]} \to \{\text{True}, \text{False}\}$.

This is sometimes called the "asymmetric" multilinear setting, is natively supported by known instantiations of graded encoding schemes [GGH13a, CLT13], and was used in previous works. Observe that in this setting, if the adversary is given a matrix entirely encoded in $G_{\{1\}}$, for example, then it will not be possible for the adversary 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. This is essential: if the adversary could compute ranks, then our goal would be impossible. Note however, that computation of such ranks is beyond the scope even of new zeroizing attacks, because it would require obtaining meaningful information from levels far "above" the top level.

Our analysis 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. For each such monomial m, we associate with it a coefficient α_m that the adversary could potentially choose arbitrarily. Finally, the adversary's polynomial is a giant sum $p = \Sigma_m \alpha_m m$ over all these potential monomials. We first 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 matrices $\{\mathbf{R}_i\}$ are randomly chosen during obfuscation. Therefore, by the Schwartz-Zippel lemma, we know that unless the adversary's

[1] 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.

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 end up being the zero polynomial over the entries of the $\mathbf{R}_i$ matrices.

Our analysis proceeds inductively. At its heart, in the inductive step, we consider what a single $\mathbf{R}_i$ matrix and its inverse $\mathbf{R}_i^{-1}$ can contribute to the adversary's polynomial. At a high level, we divide the monomials m into two categories: First, we consider all those monomials that do not arise from standard matrix multiplication. By the examining the structure of the $\mathbf{R}_i^{-1}$ matrix together with the constraint that the adversary's polynomial must be identically zero over the entries of $\mathbf{R}_i$, we are able to conclude that these monomials must have zero coefficients: $\alpha_m = 0$, because otherwise this monomial's contributions over the entries of $\mathbf{R}_i$ would survive and the adversary's polynomial cannot be identically zero over the entries of $\mathbf{R}_i$. Next, we do the same for those monomials that do arise in standard matrix multiplication; however this time we instead conclude that the coefficients of these monomials must be the same: that is, $\alpha_m = \alpha_{m'}$ for monomials m and m' that both arise in matrix multiplication.

As a result, inductively, we can 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. In other words, such an adversarial polynomial will result in an encoding of 0 only if the iterated matrix multiplication yielded the identity matrix, as desired. Even though this analysis is not efficient, as mentioned above, 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 simulate their structure at all.

Directions for future work. An obvious next step is to consider obfuscating non-evasive functions. To do so, we will need to look precisely at the kinds of elements that can be obtained using zeroizing attacks during zero testing for general (non-evasive) functions. This is beyond the scope of our paper. However, 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 elements arise in the general case.

2 Preliminaries

2.1 Evasive circuits

We define evasive circuit collections as in Barak et al. [BBC+14], 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 2.1. 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 $\mathsf{Aux} : \mathcal{C}_\ell \to \{0,1\}^*$ 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. [BGK+14] and Ananth et al. [AGIS14], with one exception: we also consider giving both the adversary and simulator an auxiliary input determined by the program.

Definition 2.2 (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 $\mathcal{M}$:

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

  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 2.3 (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 Def. 2.2. 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 Def. 2.2 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 2.4 (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 can be found in Appendix A. We describe in Appendix C how to build these branching programs from other computational models.

Definition 2.5. 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.

Definition 2.6. 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 2.7. 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 2.8. *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 [BGK+14] and [AGIS14]. 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 [GGH13a]. 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 2.9. 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 [MSW14], which modifies the straddling set system of [BGK+14] to obtain a denser intersection graph between the subsets. This extra power is used in Section 6 when showing that the adversary cannot create low-level encodings of 0.

Definition 2.10 (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 2.11. *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 Def. 2.10, 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 Def. 2.10 we must have $T = \mathbb{U}$ which contradicts our assumption. $\square$

We use the following construction from [MSW14].

Construction 2.12 (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. [AGIS14]. 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 [AGIS14], this distinction is superficial: the bookends of [AGIS14] 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 [AGIS14], 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 $(d_0, d_1, \dots, d_\ell)$, 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 [Kil88] 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

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

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 3.1 (Adapted from [AGIS14]). *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 $b_c^i = x_{\mathsf{inp}_c(i)}$ 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

$$0 = \left(\prod_{i\in[\ell]} \mathbf{D}_{i,b_0^i,b_1^i}\right)[s,t] = \beta_s\gamma_t \left(\prod_{i\in[\ell]} \alpha_{i,b_0^i,b_1^i} \mathbf{R}_{i-1}^{adj} \cdot \mathbf{B}_{i,b_0^i,b_1^i} \cdot \mathbf{R}_i\right)[s,t]$$
$$= \beta_s\gamma_t \left(\left(\prod_{i\in[\ell]} \alpha_{i,b_0^i,b_1^i}\right)\left(\prod_{i\in[\ell]} \mathbf{B}_{i,b_0^i,b_1^i}\right)\right)[s,t] = \left(\beta_s\gamma_t \prod_{i\in[\ell]} \alpha_{i,b_0^i,b_1^i}\right)(BP_{arith}(x)[s,t])$$

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 [BR13, BGK+14], or a variant of it [PST14b, AGIS14]. 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. Crucially, previous works 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. Unfortunately, this step of the analysis 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. Therefore, we need to replace the Kilian randomization theorem with a new theorem suitable in this 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 4.1. 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 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

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

Theorem 4.2. *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 $\widehat{\mathbf{A}_k} = \mathbf{R}_{k-1} \cdot \mathbf{A}_k \cdot \mathbf{R}_k^{adj}$, 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 4.2 is proved for $n-1$. Consider an arbitrary allowable polynomial p. We can write p as

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

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:

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

where

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

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:

$$\sum_i \alpha'_{i,i,\sigma(i)} A_{1,1,\sigma(i)} = 0 \text{ for all } \sigma \tag{4.1}$$

- $R_{1,m,j}\prod_{t\neq i} R_{1,\sigma(t),t}$ 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

  $$\alpha'_{j,i,\ell} = 0 \text{ for all } i, j \neq i, \ell \tag{4.2}$$

- $R_{1,m,i}\prod_{t\neq i} R_{1,\sigma(t),t}$ 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:

$$\alpha'_{i,i,\ell} = \beta_\ell \text{ for all } i, \ell \tag{4.3}$$

- $R_{1,\sigma(i),j} \prod_{t \neq i} R_{1,\sigma(t),t}$ 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 Equation 4.1 and combining with Equation 4.3, we have that

$$\sum_\ell \beta_\ell A_{1,1,\ell} = 0 \tag{4.4}$$

Now we can expand $\alpha'_{j,i,\ell}$ and β_i in Equations 4.2 and 4.4, obtaining:

$$0 = \alpha'_{i,j,\ell} = \sum_{j_2, i_3, \ldots, j_{n-1}, i_n} \alpha_{j,i,j_2,i_3,\ldots,j_{n-1},i_n} (\mathbf{A}_2 \cdot \mathbf{R}_2)_{\ell,j_2} \widehat{A}_{3,i_3,j_3} \ldots \widehat{A}_{n,i_n,1} \text{ for all } \ell, i, j \neq i \tag{4.5}$$

$$\begin{aligned} 0 = \sum_\ell \beta_\ell A_{1,1,\ell} &= \sum_{\ell, j_2, i_3, \ldots, j_{n-1}, i_n} \alpha_{i,i,j_2,i_3,\ldots,j_{n-1},i_n} A_{1,1,\ell} (\mathbf{A}_2 \cdot \mathbf{R}_2)_{\ell,j_2} \widehat{A}_{3,i_3,j_3} \ldots \widehat{A}_{n,i_n,1} \\ &= \sum_{j_2, i_3, \ldots, j_{n-1}, i_n} \alpha_{i,i,j_2,i_3,\ldots,j_{n-1},i_n} (\mathbf{A}_1 \cdot \mathbf{A}_2 \cdot \mathbf{R}_2)_{1,j_2} \widehat{A}_{3,i_3,j_3} \ldots \widehat{A}_{n,i_n,1} \text{ for all } i \end{aligned} \tag{4.6}$$

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 4.2 for $n-1$. Moreover, the right side of Equation 4.5 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

$$\alpha_{j,i,j_2,i_3,\ldots,j_{n-1},i_n} = 0 \text{ for all } j \neq i \tag{4.7}$$

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 4.2. Moreover, for each i, Equation 4.6 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 Equation 4.7, 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 4.2). Therefore, $\mathbf{A}'_2, \mathbf{A}_3, \ldots, \mathbf{A}_n$ satisfy the second set of requirements in Theorem 4.2. By induction, for each i, the polynomial in Equation 4.6 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

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

This means we can write

$$\alpha'_{j,i,\ell} = 0 \text{ for all } j \neq i \text{ (by Equation 4.7 and the definition of } \alpha'_{i,j,\ell})$$
$$\alpha'_{i,i,\ell} = \gamma_i \sum_{i_3,\ldots,i_n} (\mathbf{A}_2 \cdot \mathbf{R}_2)_{\ell,i_3} \widehat{A}_{3,i_3,i_4} \ldots \widehat{A}_{n,i_n,1} = \gamma_i \left(\mathbf{A}_2 \cdot \mathbf{A}_3 \cdots \mathbf{A}_n\right)_{\ell,1}$$

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 4.2 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 4.2 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. $\square$

5 Sketch of VBB Security Proof

We now explain how to use Theorem 4.2 to prove the VBB security of our obfuscator. The full security proof appears in Appendix D.

In this sketch, we will pay special attention to the steps in our proof that deviate from previous works [BGK+14, AGIS14]. 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 5.1. *If BP^S outputs non-shortcuttingbranching programs, then for any PPT adversary $\mathcal{A}$, there is a PPT simulator* Sim *such that*

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

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[2]. 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 [BGK+14] and [AGIS14] (and some extra analysis of our own to handle multi-bit outputs) reduces the problem of determining if p evaluates to zero to solving the following 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 4.2. Theorem 4.2 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

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

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 Section 3 is applied to an *evasive* function, any poly-time adversary will have only negligible probability in constructing an encoding of 0.

Definition 6.1. 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 6.2. *Let $\mathcal{O}$ be the obfuscator from Section 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 6.2, 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 6.3. 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 6.4. *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. [MSW14, Def. 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 2.11 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. $\square$

We now prove the main theorem of this section. The proof uses the simulator Sim of Theorem D.1 in a non-black-box way, and specifically relies on properties of the decomposition $p = \sum_x \alpha_x p_x$ given by Lemma D.3. We remark that evasive functions are inherently single-bit output, and thus we will not need the further decomposition given by Lemma D.5.

Proof of Theorem 6.2. 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 6.4 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 D.1. 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 D.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 D.1 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 4.2 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 $\mathcal{B}$:

$$\Pr\left[f\left(\mathcal{B}^f(1^\ell)\right) = 0\right] \geq \mathcal{P}'(\mathcal{A}) - \mathsf{negl}(\ell). \tag{6.1}$$

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 6.4. 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 (6.1), 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. $\square$

6.1 Bootstrapping to P/poly

In this subsection, we show that if there exists a VBB (resp. iO) obfuscator $\mathcal{O}_{\mathsf{NC}^1}$ for evasive functions computed by log-depth circuits (or more generally poly-size BPs), then there exists a VBB (resp. iO) obfuscator $\mathcal{O}_{\mathsf{P/poly}}$ for evasive functions computed by poly-size circuits (i.e. P/poly). These proofs follow previous "bootstrapping" proofs, namely from [GGH+13b] in the case of iO and from [BR14] in the case of VBB. However, we take care to ensure that the proofs go through *even under the assumption that $\mathcal{O}_{\mathsf{NC}^1}$ is only an obfuscator for evasive functions*; indeed, this is the main reason that we re-prove these theorems here.

We refer the reader to [GGH+13b, App. B] for background material on FHE, statistical simulation-sound non-interactive zero-knowledge proofs, and low-depth proofs. Briefly, a non-interactive proof is low-depth if the verifier can be implemented in NC^1.

6.1.1 Bootstrapping for VBB obfuscation

Let $\mathcal{C}$ be an evasive circuit collection on n-bit inputs, where each $C \in \mathcal{C}$ has size $|C| \leq p(n) = \mathrm{poly}(n)$. Let U be a universal circuit evaluating circuits of size $p(n)$ on n-bit inputs; in particular, $U(C,x) = C(x)$ for every $C \in \mathcal{C}$ and $x \in \{0,1\}^n$. Let $\mathcal{O}_{\mathsf{NC}^1}$ be an VBB obfuscator for all evasive circuit collections in NC^1. We denote an FHE scheme by the tuple of PPT algorithms $(\mathsf{FHE.KeyGen}, \mathsf{FHE.Enc}, \mathsf{FHE.Dec}, \mathsf{FHE.Eval})$, and we assume that $\mathsf{FHE.Dec}$ can be implemented in NC^1. The construction has two algorithms: Obfuscate and Evaluate.

Given input (m, e, ϕ), $\mathrm{P1}^{(\mathsf{SK_{FHE}}, g)}$ proceeds as follows:

- Check if ϕ is a valid low-depth proof for the NP-statement:

$$e = \mathsf{FHE.Eval}(\mathsf{PK_{FHE}}, U(\cdot, m), g).$$

- If the check fails, output 1. Else, output $\mathsf{FHE.Dec}(e, \mathsf{SK_{FHE}})$.

Figure 1

Obfuscate($1^n, C \in \mathcal{C}$):

- Generate $(\mathsf{PK_{FHE}}, \mathsf{SK_{FHE}}) \leftarrow \mathsf{FHE.KeyGen}(1^n)$. If we are using a leveled FHE scheme, the number of levels should be set to the depth of U.

- Encrypt $g = \mathsf{FHE.Enc}(\mathsf{PK}_{\mathsf{FHE}}, C)$.
- Generate $O := \mathcal{O}_{\mathsf{NC}^1}(\mathrm{P1}^{(\mathsf{SK}_{\mathsf{FHE}}, g)})$, where P1 is the NC^1 circuit described in Figure 1.
- Output the obfuscation $\sigma := (O, \mathsf{PK}_{\mathsf{FHE}}, g)$.

Evaluate($\sigma = (O, \mathsf{PK}_{\mathsf{FHE}}, g), m$): The Evaluate algorithm takes in the obfuscation output σ along with an input m, and computes $C(m)$ as follows.

- Compute $e = \mathsf{FHE.Eval}(\mathsf{PK}_{\mathsf{FHE}}, U(\cdot, m), g)$.
- Compute a low depth proof ϕ that e was computed correctly.
- Run $O(m, e, \phi)$ and output the result.

Correctness. First, we verify that the size of the circuit evaluating P1 is in NC^1. The first step of the program is in NC^1 because we apply a low-depth proof and so the the verifier can be implemented in NC^1. The next step of the program is also in NC^1 because we use an FHE scheme in which decrpytion can be done in NC^1. Hence, the whole circuit describing P1 is in NC^1.

The correctness of FHE encryption means that g will be an encryption of the circuit C. And the correctness of FHE evaluation means that e will be an encryption of $C(m)$. The correctness of $\mathcal{O}_{\mathsf{NC}^1}$ guarantees that the obfuscation of the program $\mathrm{P1}(SK_{\mathsf{FHE}}, g)$ will be executed faithfully. Thus, on a correct execution, step 1 of the program P1 will pass and the output will be $C(m)$.

Security Proof. Before proving security for the scheme described above, we observe that the function family $\{P1^{(\mathsf{SK}_{\mathsf{FHE}}, g)}\}$ defined above is evasive when $\mathcal{C}$ is, even in the presence of $(\mathsf{PK}_{\mathsf{FHE}}, g)$.

Lemma 6.5. *Let $\mathcal{P}1$ be the collection of circuits $\{\mathrm{P1}^{(\mathsf{SK}_{\mathsf{FHE}}, g)}\}$ which is indexed by $C \leftarrow \mathcal{C}$ and* $(\mathsf{PK}_{\mathsf{FHE}}, \mathsf{SK}_{\mathsf{FHE}}) \leftarrow \mathsf{FHE.KeyGen}$*, where $g := \mathsf{FHE.Enc}(\mathsf{PK}_{\mathsf{FHE}}, C)$.*

If $\mathcal{C}$ is evasive, then $\mathcal{P}1$ is evasive with auxiliary input $\mathsf{Aux}(\mathrm{P1}^{(\mathsf{SK}_{\mathsf{FHE}}, g)}) := (\mathsf{PK}_{\mathsf{FHE}}, g)$.

Proof. We first show that $\mathcal{C}$ is evasive with auxiliary input Aux. Let C' be any fixed circuit of the same size as those in $\mathcal{C}$. By the semantic security of the FHE scheme, for every $C \in \mathcal{C}$, no PPT adversary can distinguish between $(\mathsf{PK}_{\mathsf{FHE}},\ g := \mathsf{FHE.Enc}(\mathsf{PK}_{\mathsf{FHE}}, C))$ and $(\mathsf{PK}_{\mathsf{FHE}},\ g' := \mathsf{FHE.Enc}(\mathsf{PK}_{\mathsf{FHE}}, C'))$ except with negligible probability. This holds even for PPTs that know C and C', and in particular for PPTs with oracle access to C.

Let $\mathcal{B}^{(\cdot)}$ be any PPT attacking the evasiveness of $\mathcal{C}$, and let $\mathcal{D}^C$ be the PPT that, on input x, runs $\mathcal{B}^C(x)$ and outputs $C(\mathcal{B}^C(x))$. We have

$$|\Pr[\mathcal{D}^C(\mathsf{PK}_{\mathsf{FHE}}, g) = 0] - \Pr[\mathcal{D}^C(\mathsf{PK}_{\mathsf{FHE}}, g') = 0]| < \mathsf{negl}(n)$$

by semantic security. Because the distribution $(\mathsf{PK}_{\mathsf{FHE}}, g')$ is independent of the distribution on $\mathcal{D}$'s oracle, we have $\Pr[\mathcal{D}^C(\mathsf{PK}_{\mathsf{FHE}}, g') = 0] < \mathsf{negl}(n)$ by $\mathcal{C}$'s evasiveness. Thus

$$\Pr[C(\mathcal{B}^C(\mathsf{PK}_{\mathsf{FHE}}, g)) = 0] = \Pr[\mathcal{D}^C(\mathsf{PK}_{\mathsf{FHE}}, g) = 0] < \mathsf{negl}(n). \tag{6.2}$$

We now prove the lemma. Let $P := \mathrm{P1}^{(\mathsf{SK}_{\mathsf{FHE}}, g)}$ over a uniform choice of $C \leftarrow \mathcal{C}$ and $(\mathsf{PK}_{\mathsf{FHE}}, \mathsf{SK}_{\mathsf{FHE}}) \leftarrow \mathsf{FHE.KeyGen}$. Assume there exists a PPT $\mathcal{A}^{(\cdot)}$ such that

$$\Pr\left[P\left(\mathcal{A}^P(\mathsf{PK}_{\mathsf{FHE}}, g)\right) = 0\right] \geq \epsilon(n)$$

for a noticeable function $\epsilon(\cdot)$. We will show there exists another PPT $\mathcal{B}$ that contradicts (6.2).

First, observe that if $\mathcal{A}$ outputs (m, e, ϕ) such that $P(m, e, \phi) = 0$, then the input must pass the check step (first step inside $\mathrm{P1}^{(\mathsf{SK}_{\mathsf{FHE}}, g)}$), because the output of the program is 1 if the check step fails. This is a consequence of the perfect soundness of the interactive proof system we use (cf. [GGH+13b, App. B]). Further, because our FHE scheme has perfect correctness and g is an encryption of C, if (m, e, ϕ) passes the check step and $P(m, e, \phi) = 0$ then $C(m) = 0$.

Now we describe the PPT $\mathcal{B}^C$. On input $(\mathsf{PK}_{\mathsf{FHE}}, g)$, $\mathcal{B}^C$ runs $\mathcal{A}^{(\cdot)}(\mathsf{PK}_{\mathsf{FHE}}, g)$. When $\mathcal{A}$ makes an oracle query (m, e, ϕ), $\mathcal{B}$ checks if $C(m) = 0$. If so $\mathcal{B}$ halts and outputs m, otherwise it returns 1 to $\mathcal{A}$ and continues the simulation. If $\mathcal{A}$ halts with output (m, e, ϕ), $\mathcal{B}$ outputs m.

By the previous observations, $\mathcal{B}$'s simulation of $\mathcal{A}$ is perfect until $\mathcal{A}$ makes a query (m, e, ϕ) such that $P(m, e, \phi) = 0$. As we have shown, this query must satisfy $C(m) = 0$. Thus we have

$$\Pr\left[C\left(\mathcal{B}^C(\mathsf{PK}_{\mathsf{FHE}}, g)\right) = 0\right] \geq \Pr\left[P\left(\mathcal{A}^P(\mathsf{PK}_{\mathsf{FHE}}, g)\right) = 0\right] \geq \epsilon(n)$$

which contradicts (6.2). □

Theorem 6.6. *Let $\mathcal{O}_{\mathsf{NC}^1}$ be an VBB obfuscator for all NC^1 evasive circuit collections, and let FHE be a perfectly correct FHE scheme whose decryption circuit is in NC^1. Then the obfuscation scheme described above is a VBB obfuscator for all poly-size evasive circuit collections.*

Proof. The proof follows [BR14, Lemma 4.3]. Let $\mathcal{O}_{\mathsf{P/poly}}$ denote the **Obfuscate** algorithm described above. Let $\mathcal{C}$ be any poly-size evasive circuit collection, and let $C \leftarrow \mathcal{C}$ be chosen uniformly. For any adversary $\mathcal{A}$ taking inputs of the form $(O, \mathsf{PK}_{\mathsf{FHE}}, g) = \mathcal{O}_P(C)$, we define the simulator $\mathsf{Sim}_P^C(1^n)$ with oracle access to C as follows:

- Generate $(\mathsf{PK}_{\mathsf{FHE}}, \mathsf{SK}_{\mathsf{FHE}}) \leftarrow \mathsf{FHE.KeyGen}(1^n)$.
- Generate g' as an encryption under $\mathsf{PK}_{\mathsf{FHE}}$ of a size-$p(n)$ circuit that always outputs 1.
- Define the program $\mathrm{P1}'$ as follows:

  Given input (m, e, ϕ), $\mathrm{P1}'$ proceeds as follows:

  – Check if ϕ is a valid low-depth proof for the NP-statement:

  $$e = \mathsf{FHE.Eval}(\mathsf{PK}_{\mathsf{FHE}}, U(\cdot, m), g').$$

  – If the check fails, output 1. Else, output $C(m)$.

  Note that the output of $\mathrm{P1}'$ can be computed by Sim_P^C as it has oracle access to C.

- Let $\mathcal{A}'$ be $\mathcal{A}$ but with its second and third inputs hardwired to $\mathsf{PK}_{\mathsf{FHE}}$ and g'. Let $\mathsf{Sim}_{\mathsf{NC}^1}^{(\cdot)}$ be the corresponding simulator for $\mathcal{O}_{\mathsf{NC}^1}$ when considered against adversary $\mathcal{A}'$. Execute $\mathsf{Sim}_{\mathsf{NC}^1}^{\mathrm{P1}'}(1^n)$ (answering oracle queries by running $\mathrm{P1}'$), and return its output.

To prove that the above simulation succeeds (i.e. that $\mathsf{Sim}_P^C(1^n)$ simulates the view of $\mathcal{A}(\mathcal{O}_P(C))$), we use a hybrid argument as described below.

- **Hybrid 1 :**
  In the first hybrid, we just run $\mathsf{Sim}_P^C(1^n)$ as described above and give its output.

- **Hybrid 2 :**
  We change g' to be the actual encryption of C rather than the encryption of a dummy circuit. The rest of the simulator works in the same manner. This hybrid is indistinguishable from the previous one because Sim_P^C does not use $\mathsf{SK_{FHE}}$, and therefore distinguishing this hybrid from the previous would contradict the semantic security of the homomorphic encryption scheme.

- **Hybrid 3 :**
  We now replace the execution of $\mathsf{Sim}_{\mathsf{NC}^1}^{(\cdot)}$ by answering its oracle queries with P1 rather than P1$'$. The verification process done by both programs, along with the correctness of $\mathsf{FHE.Eval}$ and $\mathsf{FHE.Dec}$, guarantee that the output of the simulator in this hybrid is statistically close to that in the previous hybrid.

- **Hybrid 4 :**
  We now replace the execution of $\mathsf{Sim}_{\mathsf{NC}^1}^{(\cdot)}$ with the actual execution of $\mathcal{A}'(\mathcal{O}_{\mathsf{NC}^1}(\mathrm{P1})) = \mathcal{A}(\mathcal{O}_{\mathsf{NC}^1}(\mathrm{P1}), \mathsf{PK_{FHE}}, g) = \mathcal{A}(\mathcal{O}_P(C))$. Thus this hybrid is indistinguishable from the previous by the security of $\mathcal{O}_{\mathsf{NC}^1}$. Note that here, it is crucial that P1 is an evasive function even for an adversary that knows $\mathsf{PK_{FHE}}$ and g, which is proved in Lemma 6.5.

We conclude that $\mathsf{Sim}_P^C(1^n) \equiv$ Hybrid 1 is indistinguishable from $\mathcal{A}(\mathcal{O}_P(C)) \equiv$ Hybrid 4. $\square$

6.1.2 Bootstrapping for iO

Let $\mathcal{C}$ be an evasive circuit collection on n-bit inputs, where each $C \in \mathcal{C}$ has size $|C| \le p(n) = \mathrm{poly}(n)$. Let U be a universal circuit evaluating circuits of size $p(n)$ on n-bit inputs; in particular, $U(C,x) = C(x)$ for every $C \in \mathcal{C}$ and $x \in \{0,1\}^n$. Let $\mathcal{O}_{\mathsf{NC}^1}$ be an Indistinguishability obfuscator for all evasive circuit collections in NC^1. We denote an FHE scheme by the tuple of PPT algorithms $(\mathsf{FHE.KeyGen}, \mathsf{FHE.Enc}, \mathsf{FHE.Dec}, \mathsf{FHE.Eval})$, and we assume that $\mathsf{FHE.Dec}$ can be implemented in NC^1. The construction has two algorithms: Obfuscate and Evaluate.

Obfuscate($1^n, C \in \mathcal{C}$):

- Generate $(\mathsf{PK}^1_{\mathsf{FHE}}, \mathsf{SK}^1_{\mathsf{FHE}}) \leftarrow \mathsf{FHE.KeyGen}(1^n)$ and $(\mathsf{PK}^2_{\mathsf{FHE}}, \mathsf{SK}^2_{\mathsf{FHE}}) \leftarrow \mathsf{FHE.KeyGen}(1^n)$. If we are using a leveled FHE scheme, the number of levels should be set to the depth of U.

- Encrypt $g_1 = \mathsf{FHE.Enc}(\mathsf{PK}^1_{\mathsf{FHE}}, C)$ and $g_2 = \mathsf{FHE.Enc}(\mathsf{PK}^2_{\mathsf{FHE}}, C)$.

- Generate $O := \mathcal{O}_{\mathsf{NC}^1}(\mathrm{P1}^{(\mathsf{SK}^1_{\mathsf{FHE}}, g_1, g_2)})$, where P1 is the NC^1 circuit described in Figure 2.

- Output the obfuscation $\sigma := (O, \mathsf{PK}^1_{\mathsf{FHE}}, \mathsf{PK}^2_{\mathsf{FHE}}, g_1, g_2)$.

Evaluate($\sigma = (O, \mathsf{PK}^1_{\mathsf{FHE}}, \mathsf{PK}^2_{\mathsf{FHE}}, g_1, g_2), m$): The Evaluate algorithm takes in the obfuscation output σ along with an input m, and computes $C(m)$ as follows.

- Compute $e_1 = \mathsf{FHE.Eval}(\mathsf{PK}^1_{\mathsf{FHE}}, U(\cdot, m), g_1)$ and $e_2 = \mathsf{FHE.Eval}(\mathsf{PK}^2_{\mathsf{FHE}}, U(\cdot, m), g_2)$.

- Compute a low depth proof ϕ that e_1 and e_2 were computed correctly.

- Run $O(m, e_1, e_2, \phi)$ and output the result.

Correctness: First, we verify that the size of the circuit evaluating P1 is in NC^1. The first step of the program is in NC^1 because we apply a low-depth proof and so the the verifier can be implemented in NC^1. The next step of the program is also in NC^1 because we use an FHE scheme in which decrpytion can be done in NC^1. Hence, the whole circuit describing P1 is in NC^1.

The correctness of FHE encryption means that g_1 and g_2 will be encryptions of C. The correctness of FHE evaluation means that e_1 and e_2 will be encryptions of $C(m)$. The correctness of $\mathcal{O}_{\mathsf{NC}^1}$ guarantees that the obfuscation of the program $\mathrm{P1}^{(SK^1_{\mathsf{FHE}}, g_1, g_2)}$ will be executed faithfully. Thus, on a correct execution, step 1 of the program P1 will pass and the output will be $C(m)$.

Given input (m, e_1, e_2, ϕ), $\mathrm{P1}^{(\mathsf{SK}^1_{\mathsf{FHE}}, g_1, g_2)}$ proceeds as follows:

- Check if ϕ is a valid low-depth proof for the NP-statement:

$$e_1 = \mathsf{FHE.Eval}(\mathsf{PK}^1_{\mathsf{FHE}}, U(\cdot, m), g_1) \bigwedge e_2 = \mathsf{FHE.Eval}(\mathsf{PK}^2_{\mathsf{FHE}}, U(\cdot, m), g_2).$$

- If the check fails, output 1. Else, output $\mathsf{FHE.Dec}(e_1, \mathsf{SK}^1_{\mathsf{FHE}})$.

Figure 2

Security proof. We prove that for all $C_0, C_1 \in \mathcal{C}$ there can be no polynomial time attacker $\mathcal{A}$ that wins the indistinguishability obfuscation security game (Def. 2.4) with noticeable advantage.

Theorem 6.7. *Let $\mathcal{O}_{\mathsf{NC}^1}$ be an iO obfuscator for all NC^1 evasive circuit collections, and let FHE be a perfectly correct FHE scheme whose decryption circuit is in NC^1. Then the obfuscation scheme described above is an iO obfuscator for all poly-size evasive circuit collections.*

We organize our proof into a sequence of hybrids. In the first hybrid, the challenger obfuscates C_0. We then gradually change the obfuscation in multiple hybrids into an obfuscation of C_1. We

Given input (m, e_1, e_2, ϕ), $\mathrm{P2}^{(\mathsf{SK}^2_{\mathsf{FHE}}, g_1, g_2)}$ proceeds as follows:

- Check if ϕ is a valid low-depth proof for the NP-statement:

$$e_1 = \mathsf{FHE.Eval}(\mathsf{PK}^1_{\mathsf{FHE}}, U(\cdot, m), g_1) \bigwedge e_2 = \mathsf{FHE.Eval}(\mathsf{PK}^2_{\mathsf{FHE}}, U(\cdot, m), g_2)$$

- If the check fails, output 1. Else, output $\mathsf{FHE.Dec}(e_2, \mathsf{SK}^2_{\mathsf{FHE}})$.

Figure 3

show that each successive hybrid is indistinguishable from the previous one, thus showing our obfuscator to have indistinguishability security.

- Hybrid 0:
  This hybrid corresponds to an honest execution of the indistinguishability obfuscation game where C_0 is obfuscated.

- Hybrid 1:
  Same as hybrid 0 except we now generate $g_1 = \mathsf{FHE.Enc}(\mathsf{PK}^1_{\mathsf{FHE}}, C_0)$ and $g_2 = \mathsf{FHE.Enc}(\mathsf{PK}^2_{\mathsf{FHE}}, C_1)$ Now g_1 and g_2 encrypt different circuits.

- Hybrid 2:
  We still generate $g_1 = \mathsf{FHE.Enc}(\mathsf{PK}^1_{\mathsf{FHE}}, C_0)$ and $g_2 = \mathsf{FHE.Enc}(\mathsf{PK}^2_{\mathsf{FHE}}, C_1)$ as in the previous hybrid. Now, O is generated as $\mathcal{O}_{\mathsf{NC}^1}(\mathrm{P2}^{(\mathsf{SK}^2_{\mathsf{FHE}}, g_1, g_2)})$, where P2 is defined in Figure 3.

- Hybrid 3:
  We now generate $g_1 = \mathsf{FHE.Enc}(\mathsf{PK}^1_{\mathsf{FHE}}, C_1)$ and $g_2 = \mathsf{FHE.Enc}(\mathsf{PK}^2_{\mathsf{FHE}}, C_1)$. O is still generated as $\mathcal{O}_{\mathsf{NC}^1}(\mathrm{P2}^{(\mathsf{SK}^2_{\mathsf{FHE}}, g_1, g_2)})$.

- Hybrid 4:
  We still generate $g_1 = \mathsf{FHE.Enc}(\mathsf{PK}^1_{\mathsf{FHE}}, C_1)$ and $g_2 = \mathsf{FHE.Enc}(\mathsf{PK}^2_{\mathsf{FHE}}, C_1)$ as in the previous hybrid. Now, O is generated as $\mathcal{O}_{\mathsf{NC}^1}(\mathrm{P1}^{(\mathsf{SK}^1_{\mathsf{FHE}}, g_1, g_2)})$. Thus this hybrid corresponds to an honest execution of the indistinguishability obfuscation game where C_1 is obfuscated.

We now show that each pair of adjacent hybrids are indistinguishable, which proves Theorem 6.7.

Claim 6.8. *If the FHE scheme is IND-CPA secure, then no polynomial time attacker can distinguish between Hybrid 0 and Hybrid 1 with non-negligible probability.*

Proof. We show that if there is a polynomial time attacker A that has a non-negligible difference in advantage between Hybrid 0 and Hybrid 1, then there is a polynomial time algorithm B that breaks the IND-CPA security of our FHE scheme. B begins by running A and receiving C_0,C_1.

B first gets $\mathsf{PK}^2_{\mathsf{FHE}}$ from the IND-CPA challenger. It generates $(\mathsf{PK}^1_{\mathsf{FHE}}, \mathsf{SK}^1_{\mathsf{FHE}}) \leftarrow \mathsf{FHE.KeyGen}(1^n)$. Next, it gives the IND-CPA challenger C_0, C_1 and gets back a ciphertext g. It sets $g_2 = g$. B then sets $g_1 = \mathsf{FHE.Enc}(\mathsf{PK}^1_{\mathsf{FHE}}, C_0)$ and O is created as $\mathcal{O}_{\mathsf{NC}^1}(\mathrm{P1}^{(\mathsf{SK}^1_{\mathsf{FHE}}, g_1, g_2)})$. Note that only the first secret key $\mathsf{SK}^1_{\mathsf{FHE}}$ is needed to create O and B already has this.

If the IND-CPA challenger used the first message C_0, then we are exactly in hybrid 0. If it chose the second message C_1, then we are in Hybrid 1. Therefore, if an attacker A can distinguish between the two hybrids with non-negligible advantage, it will break the IND-CPA property of the FHE scheme. □

Claim 6.9. *If the indistinguishability property holds for $\mathcal{O}_{\mathsf{NC}^1}$, then no polynomial time attacker can distinguish between Hybrid 1 and Hybrid 2.*

Proof. We show that if there is a polynomial time attacker A that has a non-negligible difference in advantage between Hybrid 1 and Hybrid 2, then there is a polynomial time algorithm B that breaks the NC^1 indistinguishability obfuscator. B begins by running A and receiving C_0, C_1.

B first generates the two FHE private keys itself, keeping both secret keys. That is, it generates $(\mathsf{PK}^1_{\mathsf{FHE}}, \mathsf{SK}^1_{\mathsf{FHE}}) \leftarrow \mathsf{FHE.KeyGen}(1^n)$ and $(\mathsf{PK}^2_{\mathsf{FHE}}, \mathsf{SK}^2_{\mathsf{FHE}}) \leftarrow \mathsf{FHE.KeyGen}(1^n)$. It then creates $g_1 =$

$\mathsf{FHE.Enc}(\mathsf{PK}^1_{\mathsf{FHE}}, C_0)$ and $g_2 = \mathsf{FHE.Enc}(\mathsf{PK}^2_{\mathsf{FHE}}, C_1)$. Next, it submits the circuits $\mathrm{P1}^{(\mathsf{SK}^1_{\mathsf{FHE}}, g_1, g_2)}$ and $\mathrm{P2}^{(\mathsf{SK}^2_{\mathsf{FHE}}, g_1, g_2)}$ to the indistinguishability obfuscator challenger. It receives back a program O' and sets $O = O'$.

Suppose that output of C_0 is equivalent to the output of C_1 on all inputs. Then both programs $\mathrm{P1}^{(\mathsf{SK}^1_{\mathsf{FHE}}, g_1, g_2)}$ and $\mathrm{P2}^{(\mathsf{SK}^2_{\mathsf{FHE}}, g_1, g_2)}$ will have the same output on all inputs. Both programs will halt and output 1 if the check does not pass. If the check does pass, this means that e_1 and e_2 are encryptions of the same message. This is due to the perfect correctness of FHE encryption and evaluation together with the fact that $U(C_0, m) = U(C_1, m)$ for all m. Since both of the programs give the same output on all inputs, we are in a valid instance of the assumption.

If the indistinguishability obfuscator used the first circuit $\mathrm{P1}^{(\mathsf{SK}^1_{\mathsf{FHE}}, g_1, g_2)}$, then we are exactly in Hybrid 1. If it chose the second circuit $\mathrm{P2}^{(\mathsf{SK}^2_{\mathsf{FHE}}, g_1, g_2)}$, then we are in Hybrid 2. Therefore, if an attacker can distinguish between the two hybrids with non-negligible advantage, it will break the indistinguishability security of $\mathcal{O}_{\mathsf{NC}^1}$. To apply $\mathcal{O}_{\mathsf{NC}^1}$ here, we note that both P1 and P2 are evasive even given $\mathsf{PK}^1_{\mathsf{FHE}}, \mathsf{PK}^2_{\mathsf{FHE}}, g_1, g_2$ by the same argument as in Lemma 6.5. □

Claim 6.10. *If the FHE scheme is IND-CPA secure, then no polynomial time attacker can distinguish between Hybrid 2 and Hybrid 3 with non-negligible probability.*

Proof. We show that if there is a polynomial time attacker A that has a non-negligible difference in advantage between Hybrid 2 and Hybrid 3, then there is a polynomial time algorithm B that breaks the IND-CPA security of our FHE scheme. B begins by running A and receiving C_0,C_1.

B first gets $\mathsf{PK}^1_{\mathsf{FHE}}$ from the IND-CPA challenger. It generates $(\mathsf{PK}^2_{\mathsf{FHE}}, \mathsf{SK}^2_{\mathsf{FHE}}) \leftarrow \mathsf{FHE.KeyGen}(1^n)$. Next, it gives the IND-CPA challenger C_0, C_1 and gets back a ciphertext g. It sets $g_1 = g$. B then sets $g_2 = \mathsf{FHE.Enc}(\mathsf{PK}^2_{\mathsf{FHE}}, C_1)$ and O is created as $\mathcal{O}_{\mathsf{NC}^1}(\mathrm{P2}^{(\mathsf{SK}^2_{\mathsf{FHE}}, g_1, g_2)})$. Note that only the second secret key $\mathsf{SK}^2_{\mathsf{FHE}}$ is needed to create O and B already has this.

If the IND-CPA challenger used the first message C_0, then we are exactly in hybrid 2. If it chose the second message C_1, then we are in Hybrid 3. Therefore, if an attacker A can distinguish between the two hybrids with non-negligible advantage, it will break the IND-CPA property of the FHE scheme. □

Claim 6.11. *If the indistinguishability property holds for $\mathcal{O}_{\mathsf{NC}^1}$, then no polynomial time attacker can distinguish between Hybrid 3 and Hybrid 4.*

Proof. We show that if there is a polynomial time attacker A that has a non-negligible difference in advantage between Hybrid 3 and Hybrid 4, then there is a polynomial time algorithm B that breaks the NC^1 indistinguishability obfuscator. B begins by running A and receiving C_0, C_1.

B first generates the two FHE private keys itself, keeping both secret keys. That is, it generates $(\mathsf{PK}^1_{\mathsf{FHE}}, \mathsf{SK}^1_{\mathsf{FHE}}) \leftarrow \mathsf{FHE.KeyGen}(1^n)$ and $(\mathsf{PK}^2_{\mathsf{FHE}}, \mathsf{SK}^2_{\mathsf{FHE}}) \leftarrow \mathsf{FHE.KeyGen}(1^n)$. It then creates $g_1 = \mathsf{FHE.Enc}(\mathsf{PK}^1_{\mathsf{FHE}}, C_1)$ and $g_2 = \mathsf{FHE.Enc}(\mathsf{PK}^2_{\mathsf{FHE}}, C_1)$. Next, it submits the circuits $\mathrm{P1}^{(\mathsf{SK}^1_{\mathsf{FHE}}, g_1, g_2)}$ and $\mathrm{P2}^{(\mathsf{SK}^2_{\mathsf{FHE}}, g_1, g_2)}$ to the indistinguishability obfuscator challenger. It receives back a program O' and sets $O = O'$.

Both programs $\mathrm{P1}^{(\mathsf{SK}^1_{\mathsf{FHE}}, g_1, g_2)}$ and $\mathrm{P2}^{(\mathsf{SK}^2_{\mathsf{FHE}}, g_1, g_2)}$ will have the same output on all inputs. This is because both programs will halt and output 1 if the check does not pass and if the check does pass, since both e_1 and e_2 are encryptions of the same message, and due to perfect correctness of FHE encryption and evaluation, both programs will output $C_1(m)$for all m. Since both of the programs give the same output on all inputs, we are in a valid instance of the assumption.

If the indistinguishability obfuscator used the circuit $\mathrm{P2}^{(\mathsf{SK}^2_{\mathsf{FHE}}, g_1, g_2)}$, then we are exactly in Hybrid 3. If it chose the circuit $\mathrm{P1}^{(\mathsf{SK}^1_{\mathsf{FHE}}, g_1, g_2)}$, then we are in Hybrid 4. Therefore, if an attacker can distinguish between the two hybrids with non-negligible advantage, it will break the indistinguishability security of the $\mathcal{O}_{\mathsf{NC}^1}$. To apply $\mathcal{O}_{\mathsf{NC}^1}$ here, we note that both P1 and P2 are evasive even given $\mathsf{PK}^1_{\mathsf{FHE}}, \mathsf{PK}^2_{\mathsf{FHE}}, g_1, g_2$ by the same argument as in Lemma 6.5. $\square$

References

[AGIS14] Prabhanjan Ananth, Divya Gupta, Yuval Ishai, and Amit Sahai. Optimizing obfuscation: Avoiding barrington's theorem. Cryptology ePrint Archive, Report 2014/222, 2014. http://eprint.iacr.org/.

[App13] Benny Applebaum. Bootstrapping obfuscators via fast pseudorandom functions. Cryptology ePrint Archive, Report 2013/699, 2013.

[Bar86] David A. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC1. In *STOC*, 1986.

[BBC+14] Boaz Barak, Nir Bitansky, Ran Canetti, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Obfuscation for evasive functions. In *Theory of Cryptography - 11th Theory of Cryptography Conference, TCC 2014, San Diego, CA, USA, February 24-26, 2014. Proceedings*, pages 26–51, 2014.

[BGK+14] Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Protecting obfuscation against algebraic attacks. In *EUROCRYPT*, 2014.

[BLR+14] Dan Boneh, Kevin Lewi, Mariana Raykova, Amit Sahai, Mark Zhandry, and Joe Zimmerman. Semantically secure order revealing encryption: Multi-input functional encryption without obfuscation, 2014.

[BR13] Zvika Brakerski and Guy N. Rothblum. Virtual black-box obfuscation for all circuits via generic graded encoding. Cryptology ePrint Archive, Report 2013/563, 2013. http://eprint.iacr.org/.

[BR14] Zvika Brakerski and Guy N. Rothblum. Virtual black-box obfuscation for all circuits via generic graded encoding. In *TCC*, pages 1–25, 2014.

[BWZ14] Dan Boneh, David J. Wu, and Joe Zimmerman. Immunizing multilinear maps against zeroizing attacks. Cryptology ePrint Archive, Report 2014/930, 2014. http://eprint.iacr.org/.

[CHL+14] Jung Hee Cheon, Kyoohyung Han, Changmin Lee, Hansol Ryu, and Damien Stehlé. Cryptanalysis of the multilinear map over the integers. Cryptology ePrint Archive, Report 2014/906, 2014. http://eprint.iacr.org/. To appear at EUROCRYPT 2015.

[Cle91] Richard Cleve. Towards optimal simulations of formulas by bounded-width programs. *computational complexity*, 1(1):91–105, 1991.

[CLT13] Jean-Sebastien Coron, Tancrede Lepoint, and Mehdi Tibouchi. Practical multilinear maps over the integers. In *CRYPTO*, pages 476–493, 2013.

[CLT14] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Cryptanalysis of two candidate fixes of multilinear maps over the integers. Cryptology ePrint Archive, Report 2014/975, 2014. http://eprint.iacr.org/.

[GGH13a] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In *EUROCRYPT*, pages 1–17, 2013.

[GGH+13b] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, pages 40–49, 2013.

[GHMS14] Craig Gentry, Shai Halevi, Hemanta K. Maji, and Amit Sahai. Zeroizing without zeroes: Cryptanalyzing multilinear maps without encodings of zero. Cryptology ePrint Archive, Report 2014/929, 2014. http://eprint.iacr.org/.

[Gie01] Oliver Giel. Branching program size is almost linear in formula size. *J. Computer and System Sciences*, pages 222–235, 2001.

[GIS+10] Vipul Goyal, Yuval Ishai, Amit Sahai, Ramarathnam Venkatesan, and Akshay Wadia. Founding cryptography on tamper-proof hardware tokens. In *TCC*, pages 308–326, 2010.

[GLSW14] Craig Gentry, Allison Lewko, Amit Sahai, and Brent Waters. Indistinguishability obfuscation from the multilinear subgroup elimination assumption. Cryptology ePrint Archive, Report 2014/309, 2014. http://eprint.iacr.org/2014/309.

[Kil88] Joe Kilian. Founding cryptography on oblivious transfer. In *STOC*, pages 20–31, 1988.

[MSW14] Eric Miles, Amit Sahai, and Mor Weiss. Protecting obfuscation against arithmetic attacks. *IACR Cryptology ePrint Archive*, 2014:878, 2014.

[PST14a] Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation from semantically-secure multilinear encodings. In *CRYPTO*, pages 500–517, 2014.

[PST14b] Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation from semantically-secure multilinear encodings. In JuanA. Garay and Rosario Gennaro, editors, *Advances in Cryptology âĂŞ CRYPTO 2014*, volume 8616 of *Lecture Notes in Computer Science*, pages 500–517. Springer Berlin Heidelberg, 2014.

[SZ14] Amit Sahai and Mark Zhandry. Obfuscating low-rank matrix branching programs. *IACR Cryptology ePrint Archive*, 2014:773, 2014.

[Zim14] Joe Zimmerman. How to obfuscate programs directly. Cryptology ePrint Archive, Report 2014/776, 2014. http://eprint.iacr.org/.

A Branching Programs

We now define branching programs. We will actually define three notions of branching programs. The first, layered (graphical) branching programs, corresponds to the standard notion of branching programs found in the literature. Second, we define the notion of a matrix branching program, which can be seen as a generalization of graphical branching programs. Finally, we define a matrix branching program sampler, which is again a generalization of matrix branching programs.

Layered Graphical Branching Programs. Our notion of a layered graphical branching program corresponds to the traditional notion of branching programs.

Definition A.1. A *(graphical) branching program* is a finite directed acyclic graph with two special nodes, a source node and a sink node, also referred to as an "accept" node. Each non-sink node is labeled with a variable x_i and can have arbitrary out-degree. Each of the out-edges is either labeled with $x_i = 0$ or $x_i = 1$. The sink node has out-degree 0. We denote a branching program by BP and denote the restriction of the branching program consistent with input x by $BP|_x$. BP accepts an input $x \in \{0,1\}^n$ if and only if there is at least one path from the source node to the accept node in $BP|_x$. The *length* ℓ of BP is the maximum length of any such path in the graph. The *node size* t of the branching program is the total number of nodes in the graph.

A *layered (graphical) branching program* is a branching program such that nodes can be partitioned into a sequence of layers L_0 through L_ℓ where all the nodes in L_{i-1} have only outgoing edges to L_i, and all of the outgoing edges from L_{i-1} are labeled with the same input variable, denoted $x_{\mathsf{inp}(i)}$ where $\mathsf{inp} : [\ell] \to [n]$. We can assume without loss of generality that L_0 contains only the source node and L_ℓ contains only the sink node. The *length* of a layered branching program is ℓ, and the *shape* $(d_0, \ldots, d_\ell)$ counts the number of nodes in each layer: $d_i = |L_i|$. Finally, the *width* w is the maximum d_i. The *node size* t of BP is still the total number of nodes in the graph, $\sum_{i=0}^{\ell} d_i$. We also define an additional quantity, the *total size* u is the sum of the products of sizes of adjacent layers $\prod_{i=1}^{\ell} d_{i-1}d_i$. Consider a slight modification to BP where there is an edge from *every* node in L_i to *every* node in L_{i+1}, and the edges are labeled with either $x_i = 0$, $x_i = 1$, $x_i = 0,1$ (representing that this edge is always used), or $x_i = \perp$ (representing that this edge is never used). Then u counts the total number of edges in BP, and therefore represents the actual size of the description of BP.

Matrix Branching Programs. Note that our definition of a matrix branching program will depart in several ways from the standard definitions of matrix branching programs in the literature. At a high level, a matrix branching program consists of a sequence of pairs of matrices $(B_{i,0}, B_{i,1})$. To evaluate the branching program on an input x, select one matrix from each pair based on the input, and multiply all of the matrices together. The matrices are shaped so that the products are valid and the end result is a scalar; otherwise, there are no restrictions on the shapes of the matrices. The branching program evaluates to 0 if and only if the product of all the matrices is 0, and otherwise it evaluates to 1. We can also easily generalize to multi-bit outputs by having the final matrix product be an actual matrix, and test each component independently for zero.

Definition A.2. A *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

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

where $\mathbf{B}_{i,b} \in \mathbb{Z}^{d_{i-1} \times d_i}$ 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

$$BP_{arith}(x) = \prod_{i=1}^{n} \mathbf{B}_{i,x_{\mathsf{inp}(i)}}$$

- $BP_{bool} : \{0,1\}^n \to \{0,1\}^{d_0 \times d_\ell}$ 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. In other words, t bounds the possible output values of BP_{arith}.

We define the width $w = \max_{i \in [0,\ell]} d_i$, node size $t = \sum_{i=0}^{\ell} d_i$, and total size $u = \sum_{i=1}^{\ell} d_{i-1} d_i$.

Fact A.3. *Any graphical layered branching program BP of length ℓ and shape $(d_0, \ldots, d_\ell)$ can be converted into a generalized matrix branching program BP' of length ℓ, shape $(d_0, \ldots, d_\ell)$, and bound $t = \prod_{i=1}^{\ell-1} d_i \leq w^{\ell-1}$ such that $BP'_{bool}(x) = BP(x)$ for all x.*

For our obfuscator, similar to existing works, we will need to actually consider *dual-input* generalized matrix branching programs:

Definition A.4. 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.

We note that is it easy to transform any normal matrix branching program into a dual input matrix branching program of the same length, shape, and bound: set $\mathsf{inp}_0 = \mathsf{inp}_1 = \mathsf{inp}$, and $\mathbf{B}_{i,b,b} = \mathbf{B}_{i,b}$ (the values $\mathbf{B}_{i,b,1-b}$ can be set arbitrarily).

Unlike previous obfuscation constructions [GGH+13b, BR13, BGK+14, PST14b, AGIS14], we allow the matrices in the branching program to be singular, and even to be rectangular. This gives us the ability to have the product matrix have any desired shape — in particular, it can be a scalar for single-bit outputs. Thus, we do not need the "bookends" used in previous works to turn the matrix product into a scalar.

We will impose one requirement on matrix branching program, called *non-shortcutting*, which will be important in the security analysis of our obfuscator. We say that a branching program *shortcuts* on an input x if there is an interval $[j,k] \subsetneq [\ell]$ strictly smaller than $[\ell]$ such that the sub-product

$$\prod_{i=j}^{k} B_{i,x_{\mathsf{inp}_0(i)},x_{\mathsf{inp}_1(i)}} = 0 \ .$$

In other words, for the input x, it is possible to determine that $BP(x)_{j,k} = 0$ prematurely without evaluating the entire product. We say that a branching program is *non-shortcutting* if it does not shortcut on any x:

Definition A.5. 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.

A similar definition holds for regular (non-dual-input) branching programs.

We will see in the next section that it is easy to convert any matrix branching program into a non-shortcutting branching program, and only increasing its width by 2.

A final property of matrix branching programs, which we call *exactness*, says that the outputs of BP_{arith} and BP_{bool} are the same on all inputs:.

Definition A.6. A matrix branching program BP is *exact* if, for all inputs x, it holds that

$$BP_{arith}(x) \in \{0,1\}^{d_0 \times d_\ell}$$

In other words,

$$BP_{arith}(x) = BP_{bool}(x)$$

In this case, we simply write $BP(x)$ to denote $BP_{airth}(x) = BP_{bool}(x) = BP_{bool(q)}$ for all $q \geq 2$.

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 A.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 A.8. *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 $BP'_{bool(q)}(x) = BP_{bool}(x)$.*

B Efficiency improvements

In all current constructions, applying a core obfuscation method directly to circuits requires overhead that grows exponentially with depth. This occurs for two reasons. First, and perhaps most problematically, the level of multilinearity required by known core obfuscators grows exponentially with the depth of the circuit being obfuscated, and known implementations of graded encoding schemes have complexity that grows polynomially with the level of multilinearity [GGH13a, CLT13]. Furthermore, the only known method for converting circuits to full-rank matrix branching programs requires the size of the representation to grow exponentially with the depth of the circuit [Bar86]. The recent construction of Zimmerman [Zim14], which operates directly on circuits, does not suffer from the latter blowup, but still requires a level of multilinearity that grows exponentially with circuit depth (using known implementations of graded encoding schemes).

Nevertheless, a core obfuscator can be used to obfuscate general (high-depth) circuits with a polynomial overhead via bootstrapping as mentioned above. However, attempting such bootstrapping for obfuscation [GGH+13b, GIS+10, App13] based on existing core obfuscators encounters overheads that are asymptotically polynomial but easily reach above 2^{100}. Such large overheads primarily arise due to the depth of the circuit that needs to be obfuscated by the core obfuscator (even 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 perhaps representing programs as circuits may not be the best approach toward efficient obfuscation. Indeed, if we can expand the classes of program representations that are amenable to direct obfuscation by our core obfuscator, then this may allow for alternative methods of bootstrapping that yield substantially better efficiency (see also [AGIS14] for a speculative discussion of one such approach). Thus, improving the capabilities of the core obfuscator is a critical goal in obfuscation research.

Our core obfuscator substantially generalizes the types of branching programs that can be directly obfuscated. In existing core obfuscators operating on branching programs [GGH+13b, BR14, BGK+14, PST14a, GLSW14, AGIS14, MSW14], the program is required to consist of square and invertible matrices. As we elaborate below, this limitation of previous work is quite restrictive. Our obfuscator does not require this condition, and instead works on *any* matrix branching program satisfying a mild natural condition called non-shortcutting[3]. Furthermore, our work achieves two

[3]Non-shortcutting is defined in Section 2.3. Intuitively, this condition requires that no intermediate product of matrices yields the all-zero matrix.

Table 1: Comparing the efficiency of obfuscation schemes for keyed formulas over different bases. We use $\tilde{O}$ to suppress the multiplicative polynomial dependence on the security parameter and other poly-logarithmic terms and O_ϵ to suppress multiplicative constants which depend on ϵ. Here s is the formula size, $\epsilon > 0$ is an arbitrarily small constant, and ϕ is a constant such that for κ-level multilinear encodings, the size of each encoding is $\tilde{O}(\kappa^\phi)$. The current best known constructions have $\phi = 2$. Evaluation time is given in the form $a \cdot b$, where a denotes the number of multilinear operations (up to lower order additive terms) and b denotes the time for carrying out one multilinear operation.

Work	Levels of Multilinearity	Size of Obfuscation/ Evaluation Time
[AGIS14] (previous work) {AND, NOT}-basis	s	$4s^3 \cdot \tilde{O}(s^\phi)$
[AGIS14] + [Gie01] (previous work) {AND, XOR, NOT}-basis	$O(s^{1+\epsilon})$	$O(2^{(2^{1/\epsilon})} s^{(1+\epsilon)}) \cdot \tilde{O}(s^{(1+\epsilon)\phi})$
This work (direct) {AND, XOR, NOT}-basis	s	$\frac{1}{4} s \log_2(s)^2 \cdot \tilde{O}(s^\phi)$

other advances over previous work:

- Our construction allows for sequences of *non-square* matrices of arbitrary compatible dimensions. We show that this flexibility yields concrete efficiency gains (see Section C for details).
- Our construction allows for a single obfuscation to yield multiple bits of output contained in different entries in the output matrix M. This is in contrast to previous work that yielded only one bit of output, and required many parallel obfuscations to obtain multiple bits.

We also show how to exploit our results to yield efficiency improvements over previous obfuscations of both Boolean formulas and layered graphical branching programs. These improvements are summarized in Tables 1 and 2.

Furthermore, our analysis can also be used in settings where obfuscation is not directly used, but where low-rank matrix branching programs are considered in the context of multilinear maps. Indeed, subsequent to the initial online publication of this work, our theorems were used by [BLR+14] to yield substantial efficiency improvements in the context of multi-input functional encryption and semantically secure order-revealing encryption.

To obtain our results for Boolean formulas, in Section C, we give a simple conversion from formulas to (low-rank) matrix branching programs that achieves qualitatively better parameters than was previously known [Cle91]. This conversion may be of independent interest.

C Building Low-Rank Branching Programs

In this section, we describe some procedures to be carried out on branching programs. We will use these procedures to show how to make any branching program non-shortcutting, and how to convert

Table 2: Comparing the efficiency of obfuscation schemes for keyed layered graphical branching programs, as defined in Section 2.3. For a layered graphical branching program, ℓ denotes the length of the branching program, and w is the width. Other notation is as in Table 1.

Work	Levels of Multilinearity	Size of Obfuscation/ Evaluation Time
[AGIS14] (previous work)	ℓ	$4\ell^3 w^2 \cdot \tilde{O}(\ell^\phi)$
This work	ℓ	$\ell w^2 \cdot \tilde{O}(\ell^\phi)$

formulas into matrix branching programs.

In particular, we will describe a way to convert any formula of size s over $\mathsf{NOT}, \mathsf{AND}, \mathsf{XOR}$ gates into an exact matrix branching program of length $s+1$ and maximum width $\lceil \log_2(s+2) \rceil$. Our result gives a qualitative improvement to a result of Cleve [Cle91], which achieves similar asymptotics, but only for balanced formula, and only for formula over $\mathsf{NOT}, \mathsf{AND}$ gates.

C.1 Operations on Generalized Matrix Branching Programs

We now describe several operations on generalized matrix branching programs. We will use these operations to build our branching program for formulas.

Transpose. Let $BP = \left(\mathsf{inp}, (\mathbf{B}_{i,0}, \mathbf{B}_{i,1})_{i \in [\ell]}\right)$ be a branching program of length ℓ and shape $(d_0, d_1, \ldots, d_\ell) \in (\mathbb{Z}^+)^{\ell+1}$. The *transpose* of BP, denoted BP^T, is a branching program of length ℓ and shape $(d_\ell, \ldots, d_0)$, given by

$$BP^T = \left(\mathsf{inp}^T, \left(\mathbf{B}^T_{\ell+1-i,0}, \mathbf{B}^T_{\ell+1-i,1}\right)_{i \in [\ell]}\right)$$

where $\mathsf{inp}^T(i) = \mathsf{inp}(\ell + 1 - i)$. Observe that $(BP^T)_{arith/bool}(x) = (BP_{arith/bool}(x))^T$. Note that if BP is exact, then so is BP^T.

Augment. Let BP be as above. The r-augmentation of BP is the branching program $BP' = \mathsf{Augment}(BP, r)$ of length ℓ and shape $(d_0 + r, d_1 + r, \ldots, d_\ell + r)$ given by

$$BP' = \left(\mathsf{inp}, \left(\mathbf{B}'_{i,0}, \mathbf{B}'_{i,1}\right)_{i \in [\ell]}\right) \text{ where } \mathbf{B}'_{i,b} = \begin{matrix} \mathbf{B}_{i,b} & \mathbf{0}^{d_{i-1} \times r} \\ \mathbf{0}^{r \times d_i} & I_r \end{matrix}$$

Observe that

$$BP'_{arith/bool}(x) = \begin{matrix} BP_{arith/bool}(x) & \mathbf{0}^{d_0 \times r} \\ \mathbf{0}^{r \times d_\ell} & I_r \end{matrix}$$

Moreover, if BP is exact, then so is BP'. We will define $\mathsf{Augment}(BP) = \mathsf{Augment}(BP, 1)$.

Linear Operations. Let BP be as above. Given a $d'_0 \times d_0$ matrix $\mathbf{L}$ and a $d_\ell \times d'_\ell$ matrix $\mathbf{R}$, we can compute the branching program $\mathbf{L} \cdot BP \cdot \mathbf{R}$ which has length ℓ, shape $(d'_0, d_1, \ldots, d_{\ell-1}, d'_\ell)$, and is given by

$$\mathbf{L} \cdot BP \cdot \mathbf{R} = \left(\mathsf{inp}, \left(\mathbf{B}'_{i,0}, \mathbf{B}'_{i,1}\right)_{i \in [ell]}\right) \text{ where } \mathbf{B}'_{i,b} = \begin{cases} \mathbf{B}_{i,b} & \text{if } i \neq 1, \ell \\ \mathbf{L} \cdot \mathbf{B}_{1,b} & \text{if } i = 1 \\ \mathbf{B}_{\ell,b} \cdot \mathbf{R} & \text{if } i = \ell \end{cases}$$

Observe that $(\mathbf{L} \cdot BP \cdot \mathbf{R})_{arith}(x) = \mathbf{L} \cdot (BP_{arith}(x) \cdot \mathbf{R})$, and that if BP is non-shortcutting, then $\mathbf{L} \cdot BP \cdot \mathbf{R}$ is also non-shortcutting.

Merge. Let $BP^{(0)}, BP^{(1)}$ be two branching programs of length $\ell^{(b)}$ and shape $(d_0^{(b)}, \dots, d_\ell^{(b)})$ for $b \in \{0, 1\}$ with the property that $d_\ell^{(0)} = d_1^{(1)}$. Then we can compute the merge of $BP^{(0)}$ and $BP^{(1)}$ as

$$BP^{(0)} \cdot BP^{(1)} = \left(\mathsf{inp}, (\mathbf{B}_{i,0}, \mathbf{B}_{i,1})_{i \in [\ell']}\right)$$

where $\ell' = \ell^{(0)} + \ell^{(1)}$, $\mathsf{inp}' : [\ell^{(0)} + \ell^{(1)}] \to [n]$ is defined as $\mathsf{inp}(i) = \begin{cases} \mathsf{inp}^{(0)}(i) & \text{if } i \leq \ell^{(0)} \\ \mathsf{inp}^{(1)}(i - \ell^{(0)}) & \text{if } i > \ell^{(1)} \end{cases}$, and $\mathbf{B}_{i,b} = \begin{cases} \mathbf{B}_{i,b}^{(0)} & \text{if } i \leq \ell^{(0)} \\ \mathbf{B}_{i-\ell^{(0)},b}^{(1)} & \text{if } i > \ell^{(0)} \end{cases}$. Observe that $\left(BP^{(0)} \cdot BP^{(1)}\right)_{arith}(x) = (BP_{arith}^{(0)}(x)) \cdot (BP_{arith}^{(1)}(x))$.

C.2 Making any branching program non-shortcutting

Let BP be an arbitrary matrix branching program of shape $(d_0, d_1, \dots, d_\ell)$. Define

$$BP' = \begin{pmatrix} I_{d_0} & 1^{d_0 \times 1} & 0^{d_0 \times 1} \end{pmatrix} \cdot \mathsf{Augment}(BP, 2) \cdot \begin{pmatrix} I_{d_\ell} \\ 0^{1 \times d_\ell} \\ 1^{1 \times d_\ell} \end{pmatrix}$$

Notice that the shape of BP' is $(d_0, d_1 + 2, \dots, d_{\ell-1} + 2, d_\ell)$. Moreover, BP'_{arith} computes the matrix

$$\begin{aligned} BP'_{arith} &= \begin{pmatrix} I_{d_0} & 1^{d_0 \times 1} & 0^{d_0 \times 1} \end{pmatrix} \cdot \begin{pmatrix} BP_{arith} & 0^{d_0 \times 2} \\ 0^{2 \times d_\ell} & I_2 \end{pmatrix} \cdot \begin{pmatrix} I_{d_\ell} \\ 0^{1 \times d_\ell} \\ 1^{1 \times d_\ell} \end{pmatrix} \\ &= \begin{pmatrix} I_{d_0} & 1^{d_0 \times 1} & 0^{d_0 \times 1} \end{pmatrix} \cdot \begin{pmatrix} BP_{arith} \\ 0^{d_\ell \times 1} \\ 1^{d_\ell \times 1} \end{pmatrix} = BP_{arith} \end{aligned}$$

Finally, we have the following:

Lemma C.1. *BP', as defined above, is non-shortcutting.*

Proof. The branching program $BP^{(L)} = \mathsf{Augment}(BP, 2) \cdot \begin{pmatrix} I_{d_\ell} \\ 0^{1 \times d_\ell} \\ 1^{1 \times d_\ell} \end{pmatrix}$ on input x computes the matrix

$$\begin{pmatrix} BP_{arith}(x) \\ 0^{d_\ell \times 1} \\ 1^{d_\ell \times 1} \end{pmatrix}$$

which always has all columns not identically zero. Therefore the sub-product of $BP^{(L)}_{arith}(x)$ consisting of all matrices except the left-most matrix will always have non-zero columns. We obtain BP' from $BP^{(L)}$ by left-multiplying the left-most matrix by a matrix. Therefore, the sub-product of $BP'_{arith}(x)$ that drops the left-most matrix is identical to the sub-product of $BP^{(L)}_{arith}(x)$ that drops the left-most matrix, and therefore has non-zero columns. Similarly, we can conclude that any sub-product that does not include the right-most matrix has non-zero rows. Therefore, the branching program is non-shortcutting, as desired. $\square$

C.3 Arithmetic Formulas to Matrix Branching Programs

We now give our conversion of formulas to matrix branching programs. We will build a a branching program for any *arithmetic* formula taking 0/1 inputs, where every gate is an arbitrary bilinear polynomial in its inputs. That is, for any such arithmetic formula f, we build a branching program BP such that $BP_{arith}(x) = f(x)$. We note, however, that BP_{bool} only reveals if $f(x)$ is zero or not.

As a first step, we build a branching program BP such that $BP_{arith}(x) = \begin{pmatrix} 1 & f(x) \end{pmatrix}$. The final branching program BP' is given as $BP' = BP \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix}$

For input wires x_i, the branching program is trivial: inp maps 1 to i, and $B_{1,b} = \begin{pmatrix} 1 & b \end{pmatrix}$.

We now build BP recursively. Suppose $f = f_0 \mathsf{OP} f_1$ for some bilinear OP. Write $y_0 \mathsf{OP} y_1 = c_0 + c_1 y_0 + c_2 y_1 + c_3 y_0 y_1$. We observe that

$$\begin{pmatrix} 1 & y_0 \end{pmatrix} \cdot \begin{pmatrix} 0 & c_2 & 1 \\ c_1 & c_3 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ y_1 & 0 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 \\ 1 & c_0 \end{pmatrix} = \begin{pmatrix} 1 & y_0 \mathsf{OP} y_1 \end{pmatrix}$$

Therefore, we can compute the branching programs BP_0 and BP_1 for f_0 and f_1, and let

$$BP = BP_0 \cdot \left(\begin{pmatrix} 0 & c_2 & 1 \\ c_1 & c_3 & 0 \end{pmatrix} \cdot \mathsf{Augment}(BP_1^T) \cdot \begin{pmatrix} 0 & 1 \\ 1 & c_0 \end{pmatrix} \right)$$

It follows that $BP_{arith}(x) = \begin{pmatrix} 1 & f(x) \end{pmatrix}$, meaning BP computes the correct function.

Now notice that f_0 and f_1 are not treated symmetrically above. Indeed, the width w of the branching program BP is equal to $\max(w_0, 1 + w_1)$, where w_0 and w_1 are the widths of BP_0 and BP_1, respectively. For very unbalanced formula, this could lead the total width to be linear in the formula size.

However, we can easily exchange the roles of f_0 and f_1. In particular, for a gate OP, let OP^T be the operation defined as $y_0 \mathsf{OP}^T y_1 = y_0 \mathsf{OP} y_1$. Now we can write $f = f_0 \mathsf{OP} f_1$ or $f = f_1 \mathsf{OP}^T f_0$. In the first case, the total width becomes $w = \max(w_0, 1 + w_1)$, and in the second case, the width becomes $w = \max(w_1, 1 + w_0)$. Therefore, we can choose which recursion to perform to arrive at the minimum:

$$w = \min\left(\max(w_0, 1 + w_1), \max(w_1, 1 + w_0)\right)$$

For reasons that make the proof below more straightforward, we actually choose the order based on the formula size, rather than the resulting branching program width. That is, we have $s_0 \geq s_1$ where s_b is the size of f_b.

Lemma C.2. *The length ℓ, width w, total nodes t and total size u of the branching program BP' satisfy*

$$\begin{aligned}\ell &= s+1\\ w &\leq \lceil \log_2(s+2)\rceil\\ t &\leq \lceil (s+1)(1+\frac{1}{2}\log_2(s+2))\rceil\\ u &\leq \lceil (s+1)(3+\log_2(s+2))^2/4\rceil\end{aligned}$$

Proof. The fact that $\ell = s+1$ follows easily from the recurrence. We will prove that the profile of BP, $(d_0,\dots,d_{s+1})$, satisfies the following:

$$\begin{aligned}d_0 = 1 \qquad d_{s+1} &= 2\\ 2 \leq d_1 &\leq \lceil \log_2(s+2)\rceil \text{ for } i\in[s]\\ \sum_{i=0}^{s+1} d_i &\leq 1+\lceil (s+1)(1+\frac{1}{2}\log_2(s+2))\rceil\\ \sum_{i=0}^{s} d_i d_{i+1} &\leq \lceil (s+1)(3+\log_2(s+2))^2/4\rceil\end{aligned}$$

The final branching program BP' has the same profile, except that d_{s+1} is set to 1 instead of 2. The lemma easy follows.

The base case where $s=0$ is trivial. For a formula $f = f_0 \mathsf{OP} f_1$ of size s, let the size of the sub-formulas f_0 and f_1 be s_0 and s_1: $s_0+s_1+1=s$. Then by induction the branching programs BP_0 and BP_1 have profiles $d_0^{(b)},\dots d_{s_b+1}^{(b)}$ for $b=0,1$ where $d_0^{(b)}=1$, $d_{s_b+1}^{(b)}=2$, and $2\leq d_i^{(b)}\leq w_b$ where $\lceil \log_2(s_b+2)\rceil$ for $i\in[1,s_b]$. Moreover, $\sum_{i=0}^{s_b+1} d_i^{(b)} \leq t_b$ where $t_b = 1+\lceil (s_b+1)(1+\frac{1}{2}\log_2(s_b+2))\rceil$ and $\sum_{i=0}^{s_b} d_i^{(b)} d_{i+1}^{(b)} \leq u_b$ where $u_b = \lceil (s+1)(3+\log_2(s+2))^2/4\rceil$

For now, suppose $s_0\geq s_1$. The case $s_1>s_0$ is handled similarly. Then BP has the profile

$$(1, d_1^{(0)}, d_2^{(0)},\dots,d_{s_0}^{(0)}, 2, d_{s_1}^{(1)}+1, d_{s_1-1}^{(1)}+1,\dots,d_1^{(1)}+1, 2)$$

If $s_0>s_1$, the total width is at most $w_0\leq\lceil\log_2(s+2)\rceil$, as desired.

Now if $s_0=s_1$, then the total width will be w_0+1. However, $s=2s_0+1$ and so

$$w_0+1 = 1+\lceil\log_2(s_0+2)\rceil = \lceil\log_2(2s_0+4)\rceil = \lceil\log_2(s+3)\rceil$$

At first, this bound looks worse than what we are trying to prove. However, we know that s is odd. Suppose $\lceil\log_2(s+3)\rceil > \lceil\log_2(s+2)\rceil$. Then it must be that $s+2$ is a power of 2, and $s+3$ is one greater. However, this is impossible since s is odd. Therefore, $w\leq\lceil\log_2(s+2)\rceil$ in this case as well.

The linear size t satisfies $t\leq t_0+t_1-2+s_1$. We have that

$$t \leq \lceil (s_0+1)(1+\frac{1}{2}\log_2(s_0+2))\rceil + \lceil (s_1+1)(1+\frac{1}{2}\log_2(s_1+2))\rceil + s_1$$

We can bound this expression as

$$t \leq 1+\lceil (s_0+1)(1+\frac{1}{2}\log_2(s_0+2)) + (s_1+1)(1+\frac{1}{2}\log_2(s_1+2)) + s_1\rceil$$

Given that $s = s_0 + s_1 + 1$ and $s_0 \geq s_1$, it is straightforward but tedious to bound this as

$$t \leq 1 + \lceil (s+1)(1 + \frac{1}{2}\log_2(s+2)) \rceil$$

Next, we observe that

$$u = \left(\sum_{i=0}^{s_0} d_i^{(0)} d_{i+1}^{(0)} \right) + 2(d_{s_1}^{(1)} + 1) + \left(\sum_{i=0}^{s_1-1} (d_i^{(1)} + 1)(d_{i+1}^{(1)} + 1) \right)$$

The term $\left(\sum_{i=0}^{s_0} d_i^{(0)} d_{i+1}^{(0)} \right)$ is equal to u_0. The term $\left(\sum_{i=0}^{s_1-1} (d_i^{(1)} + 1)(d_{i+1}^{(1)} + 1) \right)$ is equal to $\left(\sum_{i=0}^{s_1-1} d_i^{(1)} d_{i+1}^{(1)} \right) + 2 \left(\sum_{i=0}^{s_1} d_i^{(1)} \right) + s_1 - 1 - d_0^{(1)} - d_{s_1}^{(1)}$. Using the fact that $d_{s_1+1}^{(1)} = 2$, $d_0^{(1)} = 1$ and $d_i^{(1)} \geq 2$ for all other i, we can therefore, we can bound

$$u \leq u_0 + u_1 + 2t_1 + s_1 - 4$$

We can then bound this as

$$u \leq \lceil (s_0+1)(3+\log_2(s_0+2))^2/4 + (s_1+1)(3+\log_2(s_1+2))^2/4 + 2(s_1+1)(1+\frac{1}{2}\log_2(s_1+2)) + s_1 + 1 \rceil$$

It is straightforward but tedious to bound this as $u \leq \lceil (s+1)(3 + \log_2(s+2))^2/4 \rceil$, as desired. □

Making the branching program non-shortcutting. We can make BP' non-shortcutting by increasing the profile by 2 as in Lemma C.1. However, it turns out in this case it is sufficient to only increase the profile by 1. This is because the branching program BP' always has non-zero outputs, and we obtain BP' from BP by right-multiplying the rightmost matrix. Therefore, any sub-product of BP' that does not contain the rightmost matrix must be a non-zero row vector (so its one and only row is non-zero). Therefore, BP' is part-way to non-shortcutting already. However, there is still a possibility that sub-products that do contain the right-most matrix will be zero.

We will fix this by augmenting the branching program, and then collapsing it back to a scalar by modifying the left- and right-most matrices. We do this is such a way that leaving out the left-most matrix always gives a non-zero product. Our branching program is set to

$$BP^{final} = \begin{pmatrix} 1 & 0 \end{pmatrix} \cdot \mathsf{Augment}(BP') \cdot \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Lemma C.3. *BP^{final}, as constructed above, is non-shortcutting. Moreover, it satisfies*

$$\begin{aligned} w &\leq \lceil 1 + \log_2(s+2) \rceil \\ t &\leq \lceil (s+1)(4 + \log_2(s+2))/2 \rceil \\ u &\leq \lceil (s+1)(5 + \log_2(s+2))^2/4 \rceil \end{aligned}$$

Proof. By a similar analysis to Lemma C.1 and the discussion above about BP' being partially non-shortcutting, we have that BP^{final} is non-shortcutting.

The augment procedure only increases the profile by one in each coordinate except d_0 and d_{s+1}. Therefore, the bound on w follows from the previous analysis. The linear size increases by exactly $s - 1$, so the bound of t follows from the previous analysis. Finally, it is straightforward to see that the actual size u only increases by less that twice the old linear size, plus 1. Therefore, the bound on u also follows from the previous analysis. □

C.4 Extensions

Boolean Formula. Any boolean gate can be seen as a bilinear polynomial when its inputs and outputs are treated as integers. Therefore, for any boolean formula f, we can use the conversion above to build a matrix branching program BP such that $BP_{arith} = f(x)$. Since $f(x)$ is either 0 or 1, we see that $BP_{bool}(x) = BP_{arith}(x)$, and so the resulting matrix branching program is exact.

Arithmetic Formula With Integer Input. Given any arithmetic formula where the input variables are bounded in some range of size B, we can break each input variable x into $\lceil \log_2(B+1) \rceil$ input bits x_i, which can then be assembled into x using $\lceil \log_2(B+1) \rceil - 1$ gates. Thus we increase the size of the formula by approximately a factor of $\log_2 B$, but can handle large inputs. We note that while $BP_{arith}(x) = f(x)$, BP_{bool} still only reveals if $f(x)$ is 0 or not.

Arithmetic Formula With Bounded Outputs. One limitation of the above constructions is that for general arithmetic formula, the function BP_{bool} does not reveal $f(x)$, but only a single bit. When we build our obfuscators, we will see that the function BP_{bool} is what we can actually obfuscate. Therefore, we would like to build a branching program BP such that BP_{bool} reveals the entire output $f(x)$. Here we make partial progress towards this goal by solving the case where the output $f(x)$ is confined to a small range. Let f be an arithmetic formula with the guarantee that $f(x) \in [B_0, B_1]$ for some integers B_0, B_1 where $B_1 - B_0$ is relatively small. We will construct a branching program BP such that BP_{bool} (rather than BP_{arith}) reveals the entire output of f.

First, construct the branching program BP where $BP_{arith}(x) = \begin{pmatrix} 1 & f(x) \end{pmatrix}$ as above. Then construct the branching program

$$BP' = BP \cdot \begin{pmatrix} B_0 & B_0+1 & \cdots & B_1-1 & B_1 \\ -1 & -1 & \cdots & -1 & -1 \end{pmatrix}$$

Notice that

$$BP'_{arith}(x) = \begin{pmatrix} B_0 - f(x) & (B_0+1) - f(x) & \cdot & (B_1-1) - f(x) & B_1 - f(x) \end{pmatrix}$$

Then $BP'_{bool}(x)$ is all 1's, except for index $f(x) - B_0 + 2$, which will be zero. Therefore, there is a bijective mapping between $BP'_{bool}(x)$ and $f(x)$, as desired. Moreover, $BP'_{arith}(x)$ always contains a non-zero entry, so we can make the branching program non-shortcutting by only increasing the profile by 1 instead of 2.

While the above increases the maximum width to at least $B_1 - B_0 + 2$, it does not increase the total size of the branching program by much. Indeed, the above modification only increases the profile in the last position. Therefore, the vertex size increases by an additive $B_1 - B_0$, while the total size increases by at most an additive $(B_1 - B_0)w$ where w was the maximum width before the conversion. If we were restricted to using square matrices, the total size would increase by approximately $s(2(B_1 - B_0)w + (B_1 - B_0)^2)$, which is considerably worse for large $B_1 - B_0$ or large s.

D VBB Security of our Construction

We now argue the virtual black box security of our construction. Security is given by the following theorem:

Theorem D.1. *If BP^S outputs non-shortcuttingbranching programs, then for any PPT adversary $\mathcal{A}$, there is a PPT simulator* Sim *such that*

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

Proof. We construct a simulator Sim that, on input a description of an adversary $\mathcal{A}$, simulates the view of $\mathcal{A}$ on input $BP^{\mathcal{O}} = \mathcal{O}(BP^S)$, given only oracle access to BP. Sim is also given $\ell, d_0, \dots, d_\ell, \mathsf{inp}_0, \mathsf{inp}_1$.

Most steps in the simulator are identical to [AGIS14], which the exception being the simulation of zero-test queries. First, the simulator emulates the obfuscator $\mathcal{O}$ on BP. Since Sim only has oracle access to BP and thus has no way to determine the matrices B_{i,b_0,b_1}, Sim instead initializes $\mathcal{M}$ with formal variables. More precisely, Sim will maintain a table of handles and corresponding level of encodings that have been initialized so far. Sim initially creates the table with random handles corresponding to the randomized matrices C_{i,b_0,b_1}. Sim then easily emulates all of the interfaces of $\mathcal{M}$ except for zero testing. The simulator also computes the set system used for the encodings from $\mathsf{inp}_0, \mathsf{inp}_1$.

Simulating Zero-test queries. We now describe how to simulate zero-test queries by the adversary, given only oracle access to BP. Just as in [AGIS14], when the adversary submits a handle h for zero testing, Sim looks up the corresponding polynomial p in its table. As a first step, we decompose p into *single-input elements*:

Definition D.2. 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 4.1: 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 D.3 (Adapted from [BGK+14, AGIS14])**.** *The polynomial p can be efficiently decomposed into the 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 size.

The first part of Lemma D.3 follows from the decomposition in previous works. The absence of bookends, the multi-bit outputs, and the singular and rectangular matrices does not affect this part of the simulation. Further, the strong straddling set systems we use satisfy all the properties required of the (standard) straddling set systems in [BGK+14, AGIS14]. The fact that p_x are allowable is not mentioned or proved in previous works, but follows easily from the graded encoding structure.

Because we have multi-bit outputs, we will actually need to decompose the polynomials even further.

Definition D.4. A *single-input/single-output element* for an input x and output position $(s,t) \in [d_0] \times [d_\ell]$ is a polynomial $p_{x,s,t}$ whose variables are the $\widehat{\mathbf{B}_{i,x_{\mathsf{inp}_0(i)},x_{\mathsf{inp}_1(i)}}}$ matrices, and p_x is allowable in the sense, and $p_{x,s,t}$ is allowable in the sense of Definition 4.1: each monomial in the expansion of p_x contains exactly one variable from each of the $\widehat{\mathbf{B}_{i,x_{\mathsf{inp}_0(i)},x_{\mathsf{inp}_1(i)}}}$ matrices. Moreover, the variable from the matrix $\mathbf{B}_{1,\widehat{x_{\mathsf{inp}_0(1)},x_{\mathsf{inp}_1(1)}}}$ comes from row s, and the variable from the matrix $\mathbf{B}_{\ell,\widehat{x_{\mathsf{inp}_0(\ell)},x_{\mathsf{inp}_1(\ell)}}}$ comes from column t.

Lemma D.5. *Each single-input element p_x can be efficiently decomposed into a sum*

$$p_x = \sum_{s\in[d_0],t\in[d_1]} \beta_s\gamma_t p_{x,s,t}$$

where $p_{x,s,t}$ are single input/single output elements for x, s, t

Proof. Write the $\mathbf{C}$ in terms of the $\widehat{\mathbf{B}}$ and $\mathbf{S}, \mathbf{T}$, where $\mathbf{S}$ and $\mathbf{T}$ are the diagonal matrices with β_s and γ_t on the diagonal, respectively. That is,

$$\mathbf{C}_{1,b_1,b_2} = \mathbf{S}\cdot\widehat{\mathbf{B}_{1,b_1,b_2}} \qquad \mathbf{C}_{\ell,b_1,b_2} = \widehat{\mathbf{B}_{1,b_1,b_2}}\cdot\mathbf{T} \qquad \mathbf{C}_{i,b_1,b_2} = \widehat{\mathbf{B}_{i,b_1,b_2}} \text{ for each } i\in[2,\ell-1]$$

For each $s\in[d_0], t\in[d_\ell]$, set $\beta_s = 1, \gamma_t = 1$ and $\beta_{s'} = \gamma_{t'} = 0$ for all $s'\neq s, t'\neq t$. Let $p_{x,s,t}$ be the polynomial remaining. Then $p_{x,s,t}$ is exactly a single-input/single-output element for x, s, t. Moreover, after doing this for all s, t, we have that

$$p_x = \sum_{s\in[d_0],t\in[d_1]} \beta_s\gamma_t p_{x,s,t}$$

as desired. □

Next, for each $x\in D$, we query the function oracle to learn $BP(x)$, and use $BP(x)$ to determine an input distribution on which we test the various polynomials $p_{x,s,t}$. Starting at this point, our simulation and analysis departs from previous works. Existing works rely on Kilian [Kil88] simulation to argue that the distribution of test inputs matches the distribution in the actual obfuscator. This allows them to determine whether p_x should evaluate to zero or not with overwhelming probability.

Unfortunately for us, Kilian simulation only applies to square invertible matrices. Therefore, we need to modify the simulation and/or analysis to handle this.

Fix x, s, t, and let $b_c^i = x_{\mathsf{inp}_c(i)}$. For $i\in[0,\ell-1]$, let $\widehat{\mathbf{A}}_i$ denote $\widehat{\mathbf{B}_{i,b_0^i,b_1^i}}$ and $\mathbf{A}_i = \mathbf{B}_{i,b_0^i,b_1^i}$. Let $\widehat{\mathbf{A}}_1$ be row s of $\widehat{\mathbf{B}_{1,b_0^1,b_1^1}}$ and $\widehat{\mathbf{A}}_\ell$ be row t of $\widehat{\mathbf{B}_{\ell,b_0^\ell,b_1^\ell}}$[4]. Then $p_{x,s,t}$ is an allowable polynomial in the $\widehat{\mathbf{A}}_i$

For each polynomial $p_{x,s,t}$, we determine whether $p_{x,s,t}$ evaluates to zero. We do this as follows:

- If $BP(x)[s,t] = 1$, we choose totally random matrices $\widehat{\mathbf{A}}_i$, and test if $p_{x,s,t}$ evaluates to zero on these matrices. If the result is zero, we say $p_{x,s,t}$ evaluates to zero, and if the result is non-zero, we say $p_{x,s,t}$ evaluates to non-zero. There are two cases:
  - $p_{x,s,t}$ is identically zero. Then our test will give zero with probability 1, and $p_{x,s,t}$ evaluates to zero in the actual scheme with probability 1. Therefore, we correctly determine if $p_{x,s,t}$ evaluates to zero.
  - $p_{x,s,t}$ is not identically zero. Then, by Schwartz-Zippel, our test will, with overwhelming probability, obtain non-zero, and we will report non-zero. In the actual scheme, since $BP(x)[s,t] = 1$, the $\mathbf{A}_i$ satisfy the first set of requirements of Theorem 4.2. Therefore, since $p_{x,s,t}$ is allowable, Theorem 4.2 shows that $p_{x,s,t}$ is also *not* identically zero as a polynomial over the randomization matrices $\mathbf{R}_i$. Schwartz-Zippel then shows that in the actual scheme, with overwhelming probability, $p_{x,s,t}$ will evaluate to non-zero. Thus we correctly guess whether $p_{x,s,t}$ evaluates to non-zero with overwhelming probability.

[4] The simulator does not actually compute the $\mathbf{A}_i$; we are just using them for the analysis.

- If $BP(x)[s,t] = 0$, we choose random matrices $\widehat{\mathbf{A}_i}$ subject to the restriction that their product is zero, and test $p_{x,s,t}$ on these matrices. This is done as follows. Choose random $\mathbf{A}_i$ for $i \in [\ell - 1]$. Let $\mathbf{v} = (v_1, \ldots, v_{d_{\ell-1}})$ be the row vector $\prod_{i=0}^{\ell-1} \mathbf{A}_i$. Now we sample values $w_2, \ldots, w_{d_{\ell-1}}$ at random, and let $\widehat{\mathbf{A}_\ell}$ be the column vector

$$\widehat{\mathbf{A}_\ell} = \begin{pmatrix} -\sum_{i=2}^{d_{\ell-1}} v_i w_i \\ v_1 w_2 \\ v_1 w_3 \\ \vdots \\ v_1 w_{d_{\ell-1}} \end{pmatrix} \tag{D.1}$$

Then $\mathbf{v} \cdot \widehat{\mathbf{A}_\ell} = 0$. We now make the following claim:

Claim D.6. *If a polynomial p, after making the substitution in Equation D.1, becomes identically zero, then p was originally a multiple of the matrix product polynomial.*

Proof. p being identically zero in the substitution in Equation D.1 is equivalent to p being identically zero after making the substitution

$$\hat{A}_{\ell,1,1} \leftarrow \frac{-\sum_{i=2}^{d_{\ell-1}} v_i \hat{A}_{\ell,i,1}}{v_1}$$

If this substitution gives a zero polynomial, it must be that

$$\hat{A}_{\ell,1,1} + \frac{\sum_{i=2}^{d_{\ell-1}} v_i \hat{A}_{\ell,i,1}}{v_1}$$

divides p. Since p is a polynomial, we can remove the v_1 in the denominator and conclude that, in fact,

$$v_1 \hat{A}_{\ell,1,1} + \sum_{i=2}^{d_{\ell-1}} v_i \hat{A}_{\ell,i,1} = \sum_{i=1}^{d_{\ell-1}} v_i \hat{A}_{\ell,i,1}$$

divides p. But the polynomial above is exactly the matrix product polynomial, as desired. □

Now we test $p_{x,s,t}$ on the samples $\widehat{\mathbf{A}_i}$. If the result is zero, we say $p_{x,s,t}$ evaluates to zero, and if the result is non-zero, we say $p_{x,s,t}$ evaluates to non-zero. There are two cases:

  - $p_{x,s,t}$ is a multiple of the matrix product polynomial. Then our test will give zero with probability 1, and $p_{x,s,t}$ evaluates to zero in the actual scheme with probability 1. Therefore, we correctly determine if $p_{x,s,t}$ evaluates to zero.
  - $p_{x,s,t}$ is not a multiple of the matrix product polynomial. Claim D.6 then shows $p_{x,s,t}$ must be not identically zero after making the substitutions in Equation D.1. Therefore, Schwartz-Zippel shows that the polynomial evaluates to non-zero with overwhelming probability. Therefore, we will say the value is non-zero. In the actual scheme, since $BP(x)[s,t] = 0$ and BP is non-shortcutting, the $\mathbf{A}_i$ satisfy the second set of requirements for Theorem 4.2. Since $p_{x,s,t}$ is not a multiple of the matrix product polynomial but is allowable, Theorem 4.2 shows that the polynomial is not identically zero as a polynomial

over the randomization matrices $\mathbf{R}_i$. Schwartz-Zippel then shows that in the actual scheme, with overwhelming probability, $p_{x,s,t}$ will evaluate to non-zero. Thus we correctly guess whether $p_{x,s,t}$ evaluates to non-zero with overwhelming probability.

Therefore, we will correctly determine whether $p_{x,s,t}$ evaluates to 0 for each x, s, t with overwhelming probability. Now recall that

$$p = \sum_{x \in D, s \in [d_0], t \in [d_\ell]} \left(\prod_{i \in [\ell]} \alpha_{i, x_{\mathsf{inp}_0(i)}, x_{\mathsf{inp}_1(i)}} \right) \beta_s \gamma_t p_{x,s,t}$$

If any of the $p_{x,s,t}$ evaluate to non-zero, we respond to the zero-test with non-zero. If all evaluate to zero, we respond to the zero-test with zero. Since the number of $p_{x,s,t}$ is polynomial (namely $|D| \times d_0 \times d_\ell$), we can test each $p_{x,s,t}$ efficiently. In the case where any of the $p_{x,s,t}$ are non-zero, we again appeal to Schwartz-Zippel (this time on the αs, βs, and γs) to see that with overwhelming probability the polynomial p evaluates to non-zero. If all of the p_x are zero, then with probability 1 p will evaluate to zero. Therefore, we correctly guess the value of p with overwhelming probability. □