Secure obfuscation in a weak multilinear map model: A simple construction secure against all known attacks

Eric Miles* Amit Sahai* Mark Zhandry†

Abstract

All known candidate indistinguishibility obfuscation (iO) schemes rely on candidate multilinear maps. Until recently, the strongest proofs of security available for iO candidates were in a generic model that only allows "honest" use of the multilinear map. Most notably, in this model the zero-test procedure only reveals whether an encoded element is 0, and nothing more.

However, this model is inadequate: there have been several attacks on multilinear maps that exploit extra information revealed by the zero-test procedure. In particular, the authors [Crypto'16] recently gave a polynomial-time attack on several iO candidates when instantiated with the multilinear maps of Garg, Gentry, and Halevi [Eurocrypt'13], and also proposed a new "weak multilinear map model" that captures all known polynomial-time attacks on GGH13.

Subsequent to those attacks, Garg, Mukherjee, and Srinivasan [ePrint'16] gave a beautiful new candidate iO construction, using a new variant of the GGH13 multilinear map candidate, and proved its security in the weak multilinear map model assuming an explicit PRF in NC^1.

In this work, we give a simpler candidate iO construction, which can be seen as a small modification or generalization of the original iO candidate of Garg, Gentry, Halevi, Raykova, Sahai, and Waters [FOCS'13], and we prove its security in the weak multilinear map model. Our work has a number of benefits over that of GMS16.

- Our construction and analysis are simpler. In particular, the proof of our security theorem is 4 pages, versus 15 pages in GMS16.
- We do not require any change to the original GGH13 multilinear map candidate.
- We prove the security of our candidate under a more general assumption. One way that our assumption can be true is if there exists a PRF in NC^1.
- GMS16 required an explicit PRF in NC^1 to be "hard-wired" into their obfuscation candidate. In contrast, our scheme does not require any such hard-wiring. In fact, roughly speaking, our obfuscation candidate will depend only on the minimal size of such a PRF, and not on any other details of the PRF.

*UCLA and Center for Encrypted Functionalities. {enmiles,sahai}@cs.ucla.edu. Research supported in part from a DARPA/ARL SAFEWARE award, NSF Frontier Award 1413955, NSF grants 1228984, 1136174, 1118096, and 1065276, 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 ARL under Contract W911NF-15-C-0205. 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.

†MIT & Princeton. mzhandry@gmail.com. Supported in part by the Defense Advanced Research Projects Agency (DARPA) and the U.S. Army Research Office under contract number W911NF-15-C-0226.

1 Introduction

Candidates for multilinear maps [GGH13a, CLT13, GGH15, CLT15, Hal15], also called graded encoding schemes, have formed the substrate for achieving the important goal of general-purpose indistinguishability obfuscation (iO) [BGI+01, BGI+12]. Several iO candidates have appeared in the literature starting with the work of [GGH+13b]. However, all known proofs of security for candidate obfuscation schemes have relied on assumptions that are justified only in a generic multilinear group model, where, informally speaking, the adversary is limited to using the multilinear map only in an honest manner. Most notably, this model allows the adversary to submit encodings for a zero test, and in the model the adversary only learns whether the encoding is an encoding of zero or not, and nothing more.

Unfortunately this last aspect of the modeling of multilinear maps has proven extremely elusive to achieve in multilinear map candidates: zero testing seems to reveal quite a bit more than just whether an encoded element is zero or not. Indeed, all candidate constructions of multilinear maps have been shown to suffer from "zeroizing" attacks [GGH13a, CHL+15, BWZ14, CGH+15, HJ15, BGH+15, Hal15, CLR15, MF15, MSZ16] that show how to exploit the additional information leaked by zero testing to attack various schemes constructed on top of multilinear maps. In particular, a work by the authors [MSZ16] gave the first polynomial-time attack on several candidate constructions of iO [BR14, BGK+14, PST14, AGIS14, MSW14, BMSZ16] when those constructions are instantiated using the original multilinear map candidate due to Garg, Gentry, and Halevi [GGH13a]. Thus, these attacks show that our modeling of multilinear map candidates is insufficient, even as a heuristic for arguing security.

Recently the work of [BMSZ16] explicitly addressed the question of whether a weaker model of security of multilinear maps can suffice for proving the security of iO. In particular, such a model of *weak multilinear maps* must take into account known attacks on the candidate multilinear map — that is, all known polynomial-time attacks must be allowable in the model. While there are several long-standing iO candidates that are not known to be broken (see, e.g., [AJN+16, App. A]), until recently there has not been any model for justifying their security. The work of [BMSZ16] gave the first such positive result, and showed that in one such weak multilinear map model, obfuscation for evasive functions can be proven secure with only minor modifications to existing iO candidates. In [MSZ16], we posited another, more specific, weak multilinear map model that captured all known polynomial-time attacks in the context of the GGH13 multilinear map candidate. However, that work did not answer the question of whether one can construct an iO candidate for general programs that is provably secure in this model.

In a beautiful recent work, Garg, Mukherjee, and Srinivasan [GMS16] answered this question in the affirmative, showing a new, more complex, construction of an iO candidate, that is built upon a new variant of the GGH13 multilinear map candidate, whose security can be established in a weak multilinear map model which is essentially that of [MSZ16].

Our Contribution. In this work, we give a simpler candidate iO construction, which can be seen as a small modification or generalization of the original iO candidate of [GGH+13b], and we prove its security in the weak multilinear map model of [MSZ16]. Our work has a number of benefits over [GMS16]:

- Our construction and analysis are simpler. In particular, the proof of our security theorem is 4 pages, versus 15 pages in [GMS16].
- We do not require any change to the original GGH13 multilinear map candidate. In contrast, modifying the GGH13 scheme is, in the words of [GMS16, Sec. 4], "essential" for the Garg et

al. obfuscator. This is important because a significant amount of work has already gone into understanding the strengths and limitations of the original GGH13 multilinear map candidate.

- The construction in [GMS16] must be based on an explicit NC^1 Pseudo-Random Function (PRF). This explicit PRF is hard-wired into their candidate obfuscator. In contrast, our candidate obfuscator does not perform any such hard-wiring.

- We prove the security of our candidate under a more general assumption. One way that our assumption can be true is if there exists an NC^1 PRF.

- Interestingly, if our assumption is true because a PRF exists and can be computed by a matrix branching program of size $t(n)$, then our construction will only depend on this size bound $t(n)$, and not on any other details of the PRF! Indeed, our construction will just need to be padded to have size at least roughly $t(n)$, and no modification will be necessary at all if the program being obfuscated is already larger than $t(n)$.

  Philosophically, this is reminiscent of the recent work on time-lock puzzles of [BGJ+15], where their construction of a puzzle needs to be padded to have the size of some program that computes a long non-parallellizable computation. Technically, however, our methods appear to be completely unrelated.

We now give an overview of the GGH13 multilinear map candidate. Following that, we describe an objective that is common to all known polynomial-time attacks on the GGH13 multilinear map, and use this to explain the weak multilinear map model of [MSZ16]. We then outline the proof that our new candidate is secure against all known polynomial-time attacks on GGH13 (including [MSZ16]).

1.1 Overview of GGH13

For GGH13 [GGH13a] with k levels of multilinearity, the plaintext space is a quotient ring $R_g = R/gR$ where R is the ring of integers in a number field and $g \in R$ is a "small element" in that ring. The space of encodings is $R_q = R/qR$ where q is a "big integer". An instance of the scheme relies on two secret elements, the generator g itself and a uniformly random denominator $z \in R_q$. A small plaintext element α is encoded "at level one" as $u = [e/z]_q$ where e is a "small element" in the coset of α, that is $e = \alpha + gr$ for some small $r \in R$.

Addition/subtraction of encodings at the same level is just addition in R_q, and it results in an encoding of the sum at the same level, so long as the numerators do not wrap around modulo q. Similarly multiplication of elements at levels i, i' is a multiplication in R_q, and as long as the numerators do not wrap around modulo q the result is an encoding of the product at level $i + i'$.

The scheme also includes a "zero-test parameter" in order to enable testing for zero at level k. Noting that a level-k encoding of zero is of the form $u = [gr/z^k]_q$, the zero-test parameter is an element of the form $\mathbf{p}_{\mathrm{zt}} = [hz^k/g]_q$ for a "somewhat small element" $h \in R$. This lets us eliminate the z^k in the denominator and the g in the numerator by computing $[\mathbf{p}_{\mathrm{zt}} \cdot u]_q = h \cdot r$, which is much smaller than q because both h, r are small. If u is an encoding of a non-zero α, however, then multiplying by $\mathbf{p}_{\mathrm{zt}}$ leaves a term of $[h\alpha/g]_q$ which is not small. Testing for zero therefore consists of multiplying by the zero-test parameter modulo q and checking if the result is much smaller than q.

Note that above we describe the "symmetric" setting for multilinear maps where there is only one z, and its powers occur in the denominators of encodings. More generally, there is an "asymmetric" setting where there are multiple z_i.

1.2 Overview of the model

To motivate our model (which is essentially that of [MSZ16] with some clarifications), we note that all known polynomial-time attacks [GGH13a, HJ15, MSZ16] on the GGH13 graded encoding scheme share a common property. As mentioned above, these attacks work by using information leaked during zero testing. More precisely, these attacks compute a set of top-level 0-encodings via algebraic manipulations on some set of initial encodings, then apply the zero test to each top level encoding, and then perform an algebraic computation on the *results* of the zero testing to obtain an element in the ideal $\langle g \rangle$. In particular, the latter computation is agnostic to the particular value of g and to the randomization values r chosen for each initial encoding.

After obtaining a set of elements from $\langle g \rangle$, the prior attacks then use these in various different ways to mount attacks on different cryptographic constructions built on top of GGH13. However, those details are not important to us. In our model (as suggested in [MSZ16]), if the adversary succeeds in just generating an element in the ideal $\langle g \rangle$, we will say that the adversary has won.

Our model captures the type of attack described above as follows. Like the standard ideal graded encoding model, our model $\mathcal{M}$ is an oracle that maintains a table mapping generic representations called "handles" to encodings of elements $a_i \in \mathbb{Z}_p \simeq R/\langle g \rangle$. However, rather than just storing each value a_i (along with its level), we store the formal $\mathbb{Z}_p$-polynomial $a_i + g \cdot r_i$, where g is a formal variable common to all encodings and r_i is a "fresh" formal variable chosen for each a_i. Then, an adversary may use the handles to perform any set of level-respecting algebraic computations on the initial set of encodings. The result of any such computation is an encoding f which is represented as a $\mathbb{Z}_p$-polynomial in the variables g and $\{r_i\}$.

When the adversary submits a handle to a top-level encoding f for zero-testing, $\mathcal{M}$ checks whether f's constant term is 0 (which corresponds to a 0-encoding in the standard ideal model). If so, $\mathcal{M}$ returns a handle to the formal polynomial f/g (corresponding to the result of the GGH13 zero-testing procedure), and otherwise $\mathcal{M}$ responds "not zero."

Finally, the adversary may submit a post-zero-test polynomial Q of degree at most $2^{o(\lambda)}$, where throughout the paper λ is the security parameter. $\mathcal{M}$ checks whether Q, when evaluated on the set of zero-tested encodings $\{f/g\}$ the adversary has created, produces a non-zero polynomial in which every monomial is divisible by g; i.e., it checks whether Q produces a non-zero polynomial that *is* zero mod g. If so, $\mathcal{M}$ outputs "WIN", indicating that the adversary's attack was successful. Note that any such Q is an *annihilating polynomial* (Def. 4.1) for the set $\{f/g \mod g\}$.

We remark on one hypothetical attack scenario that is not captured by our model. Namely, if an adversary constructs a polynomial Q that does not annihilate the post-zero-test polynomials as formal polynomials mod g, but does still satisfy $Q(\{f/g\}) \in \langle g \rangle$ with noticeable probability over the real distribution on g and $\{r_i\}$, this is not captured by our model. However, since in GGH13 the $\{r_i\}$ are instantiated independently from a Gaussian with no relationship to g, such a Q seems implausible. We stress that no known attack works in this way.

On the degree bound. The bound $\deg(Q) \leq 2^{o(\lambda)}$ for efficient adversaries may seem somewhat artificial. Indeed, arithmetic circuits of size $\mathsf{poly}(\lambda)$ can have arbitrary exponential degree.

However, using the GGH13 graded encoding scheme, such high-degree polynomials appear difficult to compute in the non-idealized setting. This is because, in all known polynomial-time attacks on GGH13, the post-zero-test computations cannot be performed modulo the GGH13 parameter q while maintaining the correctness of the attack. Indeed, there is no modulus M known with respect to which the computations can be performed while still maintaining correctness of attacks, unless the modulus M is so large that working modulo M results in computations that are identical to the computations over $\mathbb{Z}$.

Let us explore the intuition behind why this seems to be the case. Let d be the dimension of the ring R over $\mathbb{Z}$. Recall that the goal of the attacker in our model is to recover an element of the ideal $\langle g \rangle$. In order to safely work modulo M, it needs to be the case that $M\mathbb{Z}^d$ is a sublattice of the ideal lattice $\langle g \rangle$. But g is a secret parameter of the GGH13 scheme. Until the adversary finds out something about g, it cannot be sure that any modulus M it chooses will be safe (and indeed if the computation overflows with respect to M, almost certainly any information relevant to g will be lost). But the only way we know to learn anything about g is to find an element in $\langle g \rangle$, which was the goal of the attack to begin with.

Therefore, multiplication of two elements potentially doubles the size of the elements, and an element of exponential degree will likely have exponential size. It seems difficult even to perform post-zero-test computations of *super-polynomial* degree.

At a technical level, we need to restrict to degree $2^{o(\lambda)}$ due to our use of the Schwartz-Zippel lemma, which ceases to give useful bounds when Q has larger degree. The same restriction is necessary in [GMS16].

1.3 Overview of the security proof

To explain how we prove security in this model, we first highlight the main difference between our obfuscator and previous ones. Given a matrix branching program A, our obfuscator first transforms each matrix $A_{i,b}$ into a block-diagonal matrix

$$\begin{pmatrix} A_{i,b} & \\ & B_{i,b} \end{pmatrix}$$

where each $B_{i,b}$ is uniformly random. (As mentioned above, this can be seen as a generalization of [GGH+13b], where this same block-diagonal structure was used but the $B_{i,b}$ matrix was a random diagonal matrix. Note that we choose $B_{i,b}$ to be completely random instead.) We again stress that our obfuscator does not hard-wire into it a branching program for a PRF, or for any other specific function aside from the branching program A that is being obfuscated.

Each of these block-diagonal matrices are randomized as in previous works (using Kilian [Kil88] plus independent scalars for each matrix), and encoded as in previous works using the "straddling set" level structure from [BGK+14]. The details appear in Section 3. Thus, our only deviation from the "standard recipe" for obfuscation are the random $B_{i,b}$ matrices described above.

Our proof of security shows that any polynomial Q that annihilates a set of post-zero-test encodings mod g also annihilates a poly-size set of "generic BP evaluation polynomials"

$$e_x := \beta_0 \times \prod_{i=1}^{\ell} \beta_{i,x_{\mathsf{inp}(i)}} \times \beta_{\ell+1}$$

where $\{\beta_{i,b}\}_{i\in[\ell],b\in\{0,1\}}$ (resp. $\beta_0, \beta_{\ell+1}$) are matrices (resp. vectors) of independent variables. (For details on matrix branching programs and how they are evaluated, see Section 3.)

Our main assumption (Assumption 4.3) states that annihilating a poly-size subset of $\{e_x\}_{x\in\{0,1\}^n}$ is not possible. We observe in Theorem 4.4 that, in particular, this assumption is implied by the existence of PRF in NC^1. However, we believe this assumption to be quite plausible independent of the fact that a PRF in NC^1 would imply its validity.

To show that Q annihilates a poly-size subset of $\{e_x\}_{x\in\{0,1\}^n}$, we first use the analysis of [BGK+14, BMSZ16] to decompose each top-level 0-encoding produced by the adversary into a linear combination of "honest evaluation" polynomials $f_{x_1}, \ldots, f_{x_m}$ over the obfuscated branching program, for some $\mathsf{poly}(\lambda)$-size set of inputs $x_1, \ldots, x_m$ on which the BP evaluates to 0. (Each f_{x_i}

has roughly the same form as e_x above, except that the matrix dimension, BP length, and specific input selection function inp may be different.) Thus we can view Q as a polynomial in $\{f_{x_i}/g\}_{i\in[m]}$.

For each x_i, we can write

$$f_{x_i} = f_{x_i}^{(0)} + g \cdot f_{x_i}^{(1)} + g^2 \cdot f_{x_i}^{(2)}$$

where $f_{x_i}^{(0)}$ and $f_{x_i}^{(1)}$ are polynomials over just the $\{r_i\}$ (i.e. they do not contain the variable g). Since the BP evaluates to 0 on each x_i, we can show that $f_{x_i}^{(0)}$ is the 0 polynomial, which means that $f_{x_i}/g = f_{x_i}^{(1)} + g \cdot f_{x_i}^{(2)}$. Thus by algebraic independence, if Q annihilates $\{f_{x_i}/g\}_{i\in[m]} \bmod g$, it must in particular annihilate $\{f_{x_i}^{(1)}\}_{i\in[m]}$.

We next analyze the structure of each $f_{x_i}^{(1)}$. Each can be viewed a polynomial in the entries of the original branching program A, the randomization values chosen by the obfuscator (including the $B_{i,b}$ matrices), and the randomization values r_i in the GGH13 encodings. We show that if Q annihilates $\{f_{x_i}^{(1)}\}_{i\in[m]}$, then it must also annihilate just the $B_{i,b}$ matrices. Finally, we use the Schwartz-Zippel lemma to argue that, since the $B_{i,b}$ matrices were uniformly random over a field of size $p > 2^\lambda$, Q must annihilate them *as formal polynomials*, i.e. Q annihilates the set $\{e_{x_1}, \ldots, e_{x_m}\}$.

Our proof that annihilating $\{f_{x_i}^{(1)}\}_{i\in[m]}$ implies annihilating the $B_{i,b}$ matrices uses two techniques. The first (mentioned above) is that if Q annihilates a set of polynomials $\{p_i = p_i' + u \cdot p_i''\}_i$ where the variable u appears in no p_i', then by algebraic independence Q must also annihilate $\{p_i'\}_i$. The second is that if a set of polynomials $\{q_i\}_i$ can be obtained from another set of polynomials $\{p_i\}_i$ via a change of variables, and Q annihilates $\{p_i\}_i$, then Q also annihilates $\{q_i\}_i$.

Organization. In Section 2 we formally define our model. In Section 3 we give the details of our obfuscator, and in Section 4 we give the proof of security and discuss our assumption.

2 The Model

In this section, we define our model for weak graded encoding schemes. The model is inspired by [CGH+15, App. A], and is essentially the same as the model given in [MSZ16] (a close variant of which was also used in [GMS16]), except for some details that we clarify here.

Recall that in a graded encoding scheme, there is a universe set $\mathbb{U}$, and a *value* a can be encoded at a *level* $S \subseteq \mathbb{U}$, denoted by $[a]_S$. Addition, subtraction, and multiplication of encodings are defined provided that the levels satisfy certain restrictions, as follows.

- For any $S \subseteq \mathbb{U}$: $[a_1]_S \pm [a_2]_S := [a_1 \pm a_2]_S$.
- For any $S_1, S_2 \subseteq \mathbb{U}$ such that $S_1 \cap S_2 = \emptyset$: $[a_1]_{S_1} \cdot [a_2]_{S_2} := [a_1 \cdot a_2]_{S_1 \cup S_2}$.

Further, an encoding $[a]_{\mathbb{U}}$ at level $\mathbb{U}$ can be zero-tested, which checks whether $a = 0$.

In the standard ideal graded encoding model, a stateful oracle maintains a table that maps encodings to generic representations called *handles*. Each handle explicitly specifies the encoding's level, but is independent of the encoding's value. All parties have access to these handles, and can generate new handles by querying the oracle with arithmetic operations that satisfy the above restrictions. In addition, all parties may perform a zero-test query on any handle whose level is $\mathbb{U}$, which returns a bit indicating whether the corresponding value is 0.

Our model also implements these features, but adds new features to more closely capture the power that an adversary has in the non-idealized setting. The most important new feature is that a successful zero test returns a handle to a ring element that can further be manipulated, as opposed to just returning a bit.

We now formally describe the interfaces implemented by the oracle $\mathcal{M}$ that defines our model. For concreteness, we define $\mathcal{M}$ to explicitly work over the GGH13 ring $\mathcal{R} = \mathbb{Z}[X]/(X^\eta + 1)$ and the field $\mathbb{Z}_p \simeq \mathcal{R}/\langle g \rangle$ for an appropriate $g \in \mathcal{R}$.

Initialize parameters. The first step in interacting with $\mathcal{M}$ is to initialize it with the security parameter $\lambda \in \mathbb{N}$. (Jumping ahead, this will be done by the obfuscator.) $\mathcal{M}$ defines the ring $\mathcal{R} = \mathbb{Z}[X]/(X^\eta + 1)$, where $\eta = \eta(\lambda)$ is chosen as in [GGH13a]. Then, $\mathcal{M}$ chooses $g \in \mathcal{R}$ according to the distribution in [GGH13a], and outputs the prime $p := |\mathcal{R}/\langle g \rangle| > 2^\lambda$. After initializing these parameters, $\mathcal{M}$ discards the value of g, and treats g as a *formal variable* in all subsequent steps.

Initialize elements. After the parameters have been initialized, $\mathcal{M}$ is given a universe set $\mathbb{U}$ and a set of initial elements $\{[a_i]_{S_i}\}_i$ where $a_i \in \mathbb{Z}_p$ and $S_i \subseteq \mathbb{U}$ for each i. For each initial element $[a_i]_{S_i}$, $\mathcal{M}$ defines the formal polynomial $f_i := a_i + g \cdot z_i$ over $\mathbb{Z}_p$. Here g is a formal variable that is common to all f_i, while z_i is a "fresh" formal variable[1] chosen for each f_i. Then $\mathcal{M}$ generates a handle h_i (whose representation explicitly specifies S_i but is independent of a_i), and stores the mapping "$h_i \to (f_i, S_i)$" in a table that we call the *pre-zero-test* table. Finally, $\mathcal{M}$ outputs the set of handles $\{h_i\}_i$.

Note that storing the formal polynomial f_i strictly generalizes the standard ideal model which just stores the value a_i. This is because a_i can always be recovered as the constant term of f_i, and this holds even for subsequent polynomials that are generated from the initial set via the algebraic operations defined next.

The above two initialization interfaces are each executed once, in the order listed; any attempt to execute them out of order or more than once will fail. $\mathcal{M}$ also implements the following algebraic interfaces.

Pre-zero-test arithmetic. Given two input handles h_1, h_2 and an operation $\circ \in \{+, -, \cdot\}$, $\mathcal{M}$ first locates the corresponding polynomials f_1, f_2 and level sets S_1, S_2 in the pre-zero-test table. If h_1 and h_2 do not both appear in this table, the call to $\mathcal{M}$ fails. If the expression is undefined (i.e., $S_1 \neq S_2$ for $\circ \in \{+, -\}$, or $S_1 \cap S_2 \neq \emptyset$ for $\circ \in \{\cdot\}$), the call fails. Otherwise, $\mathcal{M}$ computes the formal polynomial $f := f_1 \circ f_2$ and the level set $S := S_1 \cup S_2$, generates a new handle h, and stores the mapping "$h \to (f, S)$" in the pre-zero-test table. Finally, $\mathcal{M}$ outputs h.

Zero-testing. Given an input handle h, $\mathcal{M}$ first locates the corresponding polynomial f and level set S in the pre-zero-test table. If h does not appear in this table, or if $S \neq \mathbb{U}$, the call to $\mathcal{M}$ fails. If f's constant term is non-zero (recall that this term is an element of $\mathbb{Z}_p$), $\mathcal{M}$ outputs the string "non-zero". If instead f's constant term is 0, note that f must be divisible by the formal variable g, i.e. g appears in each of f's monomials. $\mathcal{M}$ computes the formal polynomial $f' := f/g$ over $\mathbb{Z}_p$, generates a new handle h', and stores the mapping "$h' \to f'$" in a table that we call the *post-zero-test* table. Finally, $\mathcal{M}$ outputs h'.

Post-zero-test arithmetic. Given a set of input handles $h'_1, \ldots, h'_m$ and an m-variate polynomial Q over $\mathbb{Z}$ (represented as an arithmetic circuit), $\mathcal{M}$ first locates the corresponding polynomials $f'_1, \ldots, f'_m$ in the post-zero-test table. If any h'_i does not appear in this table, the call to $\mathcal{M}$ fails.

[1] Here and for the remainder of the paper, we use z_i rather than r_i to denote the randomization values in GGH13 encodings, to avoid conflicting with the random matrices R chosen by the obfuscator. We will not need to work with the GGH13 level denominators, which were previously denoted by z_i.

Otherwise, $\mathcal{M}$ checks whether $Q(f'_1, \ldots, f'_m)$ is non-zero as a polynomial over $\mathbb{Z}_p$ which *is* zero modulo the variable g. In other words, $\mathcal{M}$ checks that $Q(f'_1, \ldots, f'_m)$ contains at least one monomial whose coefficient is not zero modulo p, and that g appears in all such non-zero monomials.[2] If this check passes, $\mathcal{M}$ outputs "WIN", otherwise it outputs $\perp$.

Definition 2.1. A (possibly randomized) adversary interacting with the model $\mathcal{M}$ is *efficient* if it runs in time $\mathsf{poly}(\lambda)$, and if each Q submitted in a post-zero-test query has degree $2^{o(\lambda)}$. Such an adversary *wins* if it ever submits a post-zero-test query that causes $\mathcal{M}$ to output "WIN".

3 The Obfuscator

Our obfuscator for matrix branching programs is closely related to that of Badrinarayanan et al. [BMSZ16]. The main difference is that, before randomizing and encoding, each matrix $A_{i,b}$ is first transformed into a block-diagonal matrix

$$\begin{pmatrix} A_{i,b} & \\ & B_{i,b} \end{pmatrix}$$

where each $B_{i,b}$ is uniformly random.

We now describe our obfuscator $\mathcal{O}$. $\mathcal{O}$ is instantiated with two parameters, $t = t(n, \lambda)$ and $s = s(n, \lambda)$, that correspond to those in Assumption 4.3.

Input. $\mathcal{O}$ takes as input a dual-input matrix branching program[3] BP of length m, width w, and input length n. Such a matrix branching program consists of an input-selection function $\mathsf{inp} : [m] \to [n] \times [n]$, $4m$ matrices $\{A_{i,b_1,b_2} \in \{0,1\}^{w \times w}\}_{i \in [m]; b_1, b_2 \in \{0,1\}}$, and two "bookend" vectors $A_0 \in \{0,1\}^{1 \times w}$ and $A_{m+1} \in \{0,1\}^{w \times 1}$. BP is evaluated on input $x \in \{0,1\}^n$ by checking whether

$$A_0 \times \prod_{i \in [m]} A_{i,x(i)} \times A_{m+1}$$

is zero or non-zero, where we abbreviate $x(i) := (x_{\mathsf{inp}(i)_1}, x_{\mathsf{inp}(i)_2})$. We make three requirements on BP (cf. [BMSZ16, Sec. 3]).

1. It is forward non-shortcutting, defined below.
2. For each $i \in [m] : \mathsf{inp}(i)_1 \neq \mathsf{inp}(i)_2$.
3. For each pair $j \neq k \in [n]$, there exists $i \in [m]$ such that $\mathsf{inp}(i) \in \{(j,k), (k,j)\}$.

Definition 3.1 ([BMSZ16])**.** A branching program $A_0, \{A_{i,b}\}_{i \in [\ell], b \in \{0,1\}}, A_{\ell+1}$ is *forward (resp. reverse) non-shortcutting* if, for every input x, the vector

$$A_0 \times \prod_{i \in [\ell]} A_{i,x(i)} \qquad \left(\text{resp.} \qquad \prod_{i \in [\ell]} A_{i,x(i)} \times A_{\ell+1} \right)$$

is non-zero. It is *non-shortcutting* if it is both forward and reverse non-shortcutting.

[2] Note that this corresponds to finding a non-trivial element in the ideal $\langle g \rangle$.

[3] These can be constructed from any NC^1 formula with $m = \mathsf{poly}(n)$ and $w = 5$ by Barrington's theorem [Bar86]. Obfuscating NC^1 formulas is sufficient to obfuscate all polynomial-size circuits [GGH+13b, BR14, App14].

Step 0: Initialize model. $\mathcal{O}$ first sends the security parameter λ to the model $\mathcal{M}$, and receives back a prime p.

Step 1: Pad BP. $\mathcal{O}$'s first modification to BP is to pad it with identity matrices (if necessary) so that it contains a set of t layers $i_1 < \ldots < i_t$ such that $(\mathsf{inp}(i_1)_1, \ldots, \mathsf{inp}(i_t)_1)$ cycles t/n times through $[n]$. This choice of inp is specifically to allow a branching program of the form in Assumption 4.3 to be transformed into one with input selection function $\mathsf{inp}(\cdot)_1$. We use $\ell \leq t + m$ to denote the length of the padded branching program.

Step 2: Extend matrices. Next, $\mathcal{O}$ extends the matrices as mentioned above. To do this, it selects 4ℓ uniformly random matrices $\{B_{i,b_1,b_2} \in \mathbb{Z}_p^{s\times s}\}_{i\in[\ell];b_1,b_2\in\{0,1\}}$ and one uniformly random vector $B_{\ell+1} \in \mathbb{Z}_p^{s\times 1}$, and defines the following matrices and vectors.

$$A'_0 := (A_0 \quad 0^s) \qquad A'_{i,b_1,b_2} := \begin{pmatrix} A_{i,b_1,b_2} & \\ & B_{i,b_1,b_2} \end{pmatrix} \qquad A'_{\ell+1} := \begin{pmatrix} A_{\ell+1} \\ B_{\ell+1} \end{pmatrix}$$

Note that this satisfies

$$A'_0 \times \prod_{i\in[\ell]} A'_{i,x(i)} \times A'_{\ell+1} \;=\; A_0 \times \prod_{i\in[\ell]} A_{i,x(i)} \times A_{\ell+1}$$

for every input $x \in \{0,1\}^n$.

Step 3: Randomize. Next, $\mathcal{O}$ generates uniformly random non-singular matrices $\{R_i\}_{i\in[\ell+1]}$ and uniformly random non-zero scalars $\alpha_0, \{\alpha_{i,b_1,b_2}\}_{i\in[\ell];b_1,b_2\in\{0,1\}}, \alpha_{\ell+1}$. Then it computes the randomized branching program, denoted $\widehat{BP}$, as follows.

$$\widehat{A_0} := \alpha_0 A'_0 \times R_1^{adj} \qquad \widehat{A_{i,b_1,b_2}} := \alpha_{i,b_1,b_2} R_i \times A'_{i,b_1,b_2} \times R_{i+1}^{adj} \qquad \widehat{A_{\ell+1}} := \alpha_{\ell+1} R_{\ell+1} \times A'_{\ell+1}$$

Here R_i^{adj} denotes the adjugate matrix of R_i that satisfies $R_i^{adj} \times R_i = \det(R_i) \cdot I$. It is easy to see that $\widehat{BP}$ computes the same function as BP, i.e.

$$\widehat{A_0} \times \prod_{i\in[\ell]} \widehat{A_{i,x(i)}} \times \widehat{A_{\ell+1}} = 0 \qquad \Leftrightarrow \qquad A_0 \times \prod_{i\in[\ell]} A_{i,x(i)} \times A_{\ell+1} = 0$$

for every input $x \in \{0,1\}^n$.

Step 4: Encode. Finally, $\mathcal{O}$ initializes $\mathcal{M}$ with the elements of the $\widehat{A}$ matrices. To do this, it uses the level structure in [BGK+14] constructed from so-called *straddling sets*. We defer the details to Appendix A, but we remark that this level structure has the property that each "honest evaluation" $\widehat{BP}(x) = \widehat{A_0} \times \prod_i \widehat{A_{i,x(i)}} \times \widehat{A_{\ell+1}}$ results in an encoding at level $\mathbb{U}$. This, in combination with the zero-test procedure, allows the obfuscated program to be evaluated.

$\mathcal{M}$'s pre-zero-test table can now be viewed as containing $Y_0, \{Y_{i,b_1,b_2}\}_{i\in[\ell];b_1,b_2\in\{0,1\}}, Y_{\ell+1}$ of the following form.

$$Y_0 = \widehat{A_0} + gZ_0 \qquad Y_{i,b_1,b_2} = \widehat{A_{i,b_1,b_2}} + gZ_{i,b_1,b_2} \qquad Y_{\ell+1} = \widehat{A_{\ell+1}} + gZ_{\ell+1}$$

Here g is a formal variable and each Z matrix is a matrix of formal variables, while the $\widehat{A}$ matrices contain $\mathbb{Z}_p$-elements.

The final branching program $\widehat{BP} = \mathcal{O}(BP)$ has length ℓ (satisfying $t \leq \ell \leq m + t$) and width $w + s$. In the proof of Theorem 4.5, we will use the fact that any branching program of the form in Assumption 4.3 can be transformed (by padding with identity matrices) into one with length ℓ whose input selection function is the same as $\mathsf{inp}(\cdot)_1$.

Definition 3.2. $\mathcal{O}$ is *secure* in the model $\mathcal{M}$ of Section 2 if, for every BP matching $\mathcal{O}$'s input specification and every efficient adversary $\mathcal{A}$ interacting with $\mathcal{M}$, $\Pr[\mathcal{A} \text{ wins}] < \mathsf{negl}(\lambda)$ when $\mathcal{M}$ is initialized by $\mathcal{O}(BP)$. (Here the probability is over the randomness of $\mathcal{O}$ and $\mathcal{A}$.)

4 Security of Our Obfuscator

We first state two definitions, and then state the assumption under which we will prove security. After that, we prove our security theorem.

Definition 4.1. Let $f_1, \ldots, f_m$ be a set of polynomials over some common set of variables. Then an m-variate polynomial Q *annihilates* $\{f_i\}_{i \in [m]}$ if $Q(f_1, \ldots, f_m)$ is zero as a formal polynomial.

Definition 4.2. A matrix branching program BP is *L-bounded* for $L \in \mathbb{N}$ if every intermediate value computed when evaluating BP on any input is at most L. In particular all of BP's outputs and matrix entries are $\leq L$.

Our assumption essentially states that no efficiently computable polynomial can annihilate *every* branching program's evaluation polynomials on some efficiently computable set of inputs. (The assumption is parameterized by the length t and width s of the branching program.) In the assumption, we implicitly use a more general notion of how a branching program computes a function than was used in the previous section. Namely, the function computed can have range $[2^\lambda]$ (rather than $\{0,1\}$) by taking the output to be the value resulting from multiplying the appropriate vectors and matrices (rather than a bit indicating whether this value is 0).

Assumption 4.3 (The (t, s)-branching program un-annihilatability (BPUA) assumption)**.** *Let $t = \mathsf{poly}(n, \lambda)$ and $s = \mathsf{poly}(n, \lambda)$ be parameters. Let $\mathcal{A}$ denote a PPT that, on input $(1^n, 1^\lambda)$, outputs a $\mathsf{poly}(\lambda)$-size set $\mathcal{X} \subseteq \{0,1\}^n$ and a $\mathsf{poly}(\lambda)$-size, $2^{o(\lambda)}$-degree polynomial Q over $\mathbb{Z}$.*

For sufficiently large n and λ, all primes $2^\lambda < p \leq 2^{\mathsf{poly}(\lambda)}$, and all such $\mathcal{A}$, there exists a (single-input) 2^λ-bounded matrix branching program $BP : \{0,1\}^n \to [2^\lambda]$ of length t and width s, whose input selection function iterates over the n input bits t/n times, such that

$$\Pr\left[Q\left(\{BP(x)\}_{x \in \mathcal{X}}\right) = 0 \pmod{p}\right] < \mathsf{negl}(\lambda)$$

where the probability is over $\mathcal{A}$'s randomness.

We observe that Assumption 4.3 is in particular implied by the existence of PRF in NC^1 secure against P/poly (with t, s related to the size of such PRF).

Theorem 4.4. *Let t and s be as in Assumption 4.3. If there exists a PRF $F_k : \{0,1\}^n \to [2^\lambda]$ that*

- *is computable by a length-t/n, width-s, 2^λ-bounded matrix branching program BP_k, and*
- *is secure against non-uniform, polynomial-time adversaries (i.e. secure against P/poly)*

then Assumption 4.3 holds.

Note that we take BP_k's matrix entries to be computed as a function of the PRF key k.

Proof. Assume that Assumption 4.3 is false, and fix a PPT $\mathcal{A}$ and a prime p such that

$$\Pr\left[Q\left(\{BP(x)\}_{x\in\mathcal{X}}\right) = 0 \pmod p\right] \geq 1/\mathsf{poly}(\lambda)$$

for every BP of the form in Assumption 4.3. We give a PPT $\mathcal{A}'$ with oracle access to O that distinguishes with probability $\geq 1/\mathsf{poly}(\lambda)$ whether O implements BP_k for a uniform k or implements a uniform function $F : \{0,1\}^n \to [2^\lambda]$. We note that hardwiring p into $\mathcal{A}'$ is the only place where non-uniformity is needed.

$\mathcal{A}'$ simply runs $\mathcal{A}$ to get Q and $\mathcal{X}$, and computes $d := Q(O(x)_{x\in\mathcal{X}}) \pmod p$. Note that $\mathcal{A}'$ runs in time $\mathsf{poly}(\lambda)$ because Q and p both have this size. If O implements BP_k, then $d = 0$ with probability $\geq 1/\mathsf{poly}(\lambda)$. To see this, note that BP_k can be transformed (by padding with identity matrices) into an equivalent branching program of the form in Assumption 4.3 due to the input selection function there.

On the other hand, if O implements a random function, then since $p > 2^\lambda$ and $\deg(Q) = 2^{o(\lambda)}$, $d = 0$ with probability $< \mathsf{negl}(\lambda)$ by the Schwartz-Zippel lemma. □

For further discussion on our assumption, including the plausibility of PRF necessary for Theorem 4.4, see Section 4.2.

4.1 Our Main Theorem

Theorem 4.5. *Let $\mathcal{O}$ be the obfuscator from Section 3 with parameters t and s. If the (t,s)-BPUA assumption holds, $\mathcal{O}$ is secure in the model $\mathcal{M}$ of Section 2.*

We note that this theorem also implies that $\mathcal{O}$ achieves VBB security in the model from Section 2. To see this, first note that the initialization, pre-zero-test arithmetic, and zero-test interfaces can be simulated with error $\mathsf{negl}(\lambda)$ exactly as in the proof of [BMSZ16, Thm. 5.1]. Further, a simulator can simply respond to every post-zero-test query with $\perp$, and the additional error introduced by this is bounded by $\mathsf{negl}(\lambda)$ due to Theorem 4.5.

Proof. Fix a PPT adversary $\mathcal{A}$ and assume for contradiction that, with probability $\epsilon \geq 1/\mathsf{poly}(\lambda)$, $\mathcal{A}$ obtains a set of valid post-zero-test handles $h'_1, \ldots, h'_m$ and constructs a size-$\mathsf{poly}(\lambda)$, degree-$2^{o(\lambda)}$, m-variate polynomial Q over $\mathbb{Z}$ such that the post-zero-test query $(Q, h'_1, \ldots, h'_m)$ causes $\mathcal{M}$ to output "WIN". By the definition of $\mathcal{M}$, each handle h'_j must then correspond to a polynomial f'_j such that $f_j := g \cdot f'_j$ is a level-$\mathbb{U}$ polynomial in $\mathcal{M}$'s pre-zero-test table with constant term 0.

Recall that $\mathcal{M}$ is initialized with the set of $\mathbb{Z}_p$ values $\{a_i\}_i$ from the branching program $\widehat{BP}$ created by $\mathcal{O}(BP)$, and that for each such value $\mathcal{M}$ stores a polynomial $a_i + g \cdot z_i$ with formal variables g, z_i. Thus each f_j is a $\mathbb{Z}_p$-polynomial with variables $g, \{z_i\}_i$. In the following, we use $\overline{f_j}$ to denote the polynomial over the set of $\mathcal{M}$'s initial elements such that $\overline{f_j}(\{a_i + g \cdot z_i\}_i) = f_j$.

Decomposing $\overline{f_j}$. For any input x, let $\overline{f_x}$ denote the matrix product polynomial that corresponds to evaluating $\widehat{BP}(x)$, and note that $\overline{f_x}(\{a_i\}_i) = 0 \pmod p \Leftrightarrow \widehat{BP}(x) = 0 \Leftrightarrow BP(x) = 0$. The results of [BGK+14, BMSZ16] (summarized in Lemma 4.7 following this proof) show that, with probability $1 - \mathsf{negl}(\lambda)$ over the randomness of $\mathcal{O}$, for each $j \in [m]$ there is a $\mathsf{poly}(\lambda)$-size set $\mathcal{X}_j$ such that: (1) $\overline{f_j}$ is a linear combination of the polynomials $\{\overline{f_x}\}_{x\in\mathcal{X}_j}$, and (2) $BP(x) = 0$ for every $x \in \mathcal{X}_j$. (Note that the conditions of the lemma are satisfied, as we can assume wlog that the post-zero-test query we are analyzing is the first to which $\mathcal{M}$ has responded with "WIN".)

The set $\mathcal{X}_j$ and the coefficients in the linear combination depend only on the *structure* of $\overline{f_j}$, and not on $\mathcal{O}$'s randomness. So, more precisely, Lemma 4.7 says that if $\overline{f_j}$ is *not* a linear combination

of $\{\overline{f_x}\}_{x\in\mathcal{X}_j}$ for some $\mathcal{X}_j$ that satisfies $\bigwedge_{x\in\mathcal{X}_j}(BP(x)=0)$, then $f_j=\overline{f_j}(\{a_i+g\cdot z_i\}_i)$ has constant term 0 with probability $<\mathsf{negl}(\lambda)$ over the randomness of $\mathcal{O}$. Thus, we condition on the event that each $\overline{f_j}$ is decomposable in this way, which has probability $1-\mathsf{negl}(\lambda)$.

Structure of $\overline{f_x}$. Let $\mathcal{X}:=\bigcup_{j\in[m]}\mathcal{X}_j$, and consider the polynomial $f_x:=\overline{f_x}(\{a_i+g\cdot z_i\}_i)$ for any $x\in\mathcal{X}$. This is a $\mathbb{Z}_p$-polynomial with variables $g,\{z_i\}_i$, so we can "stratify" by g, writing

$$f_x=f_x^{(0)}+g\cdot f_x^{(1)}+g^2\cdot f_x^{(2)} \tag{4.1}$$

where g does not appear in the polynomials $f_x^{(0)}$ and $f_x^{(1)}$, i.e. they are polynomials in just the variables $\{z_i\}_i$. From the analysis above, we know that $f_x^{(0)}$ is the identically 0 polynomial; if not, we would not have $\overline{f_x}(\{a_i\}_i)=0 \pmod p$, and thus would not have $BP(x)=0$. So, we can write

$$f_x/g=f_x^{(1)}+g\cdot f_x^{(2)}. \tag{4.2}$$

The fact that the post-zero-test query $(Q,h_1',\ldots,h_m')$ causes $\mathcal{M}$ to output "WIN" implies that $Q(f_1',\ldots,f_m')=Q(f_1/g,\ldots,f_m/g)$ is not identically zero as a polynomial in variables g and $\{z_i\}_i$, but is identically zero modulo the variable g. Let L_j denote the linear polynomial such that $\overline{f_j}=L_j(\{\overline{f_x}\}_{x\in\mathcal{X}_j})$. Then for each $j\in[m]$, we can write

$$f_j=\overline{f_j}\left(\{a_i+g\cdot z_i\}_i\right)=L_j\left(\left\{\overline{f_x}\left(\{a_i+g\cdot z_i\}\right)_i\right\}_{x\in\mathcal{X}_j}\right)=L_j\left(\{f_x\}_{x\in\mathcal{X}_j}\right).$$

Since each L_j is linear, we then obtain an $|\mathcal{X}|$-variate polynomial Q', with $\deg(Q')=\deg(Q)$, such that $Q'(\{f_x/g\}_{x\in\mathcal{X}})=Q(\{f_j/g\}_{j\in[m]})$. Then, using (4.2) and the fact that $Q(\{f_j/g\}_{j\in[m]})$ is identically zero modulo the variable g, we must have that $Q'(\{f_x^{(1)}\}_{x\in\mathcal{X}})$ is the identically zero polynomial. In other words, Q' annihilates the set of polynomials $\{f_x^{(1)}\}_{x\in\mathcal{X}}$.

We now analyze the structure of the $f_x^{(1)}$ to show that such a Q' violates the (t,s)-BPUA assumption, which will complete the proof.

Structure of $f_x^{(1)}$. Recalling the notation from Section 3, each $\overline{f_x}$ is a polynomial in the entries of $Y_0,\{Y_{i,b_1,b_2}\}_{i\in[\ell];b_1,b_2\in\{0,1\}},Y_{\ell+1}$. Specifically, it is the polynomial

$$\overline{f_x}=Y_0\times\prod_{i\in[\ell]}Y_{i,x(i)}\times Y_{\ell+1}$$

where we abbreviate $x(i):=(x_{\mathsf{inp}(i)_1},x_{\mathsf{inp}(i)_2})$. Notice that f_x is the polynomial obtained from $\overline{f_x}$ after making the following substitution.

$$Y_0=\widehat{A_0}+gZ_0 \qquad Y_{i,b_1,b_2}=\widehat{A_{i,b_1,b_2}}+gZ_{i,b_1,b_2} \qquad Y_{\ell+1}=\widehat{A_{\ell+1}}+gZ_{\ell+1}$$

Then, because $f_x^{(1)}$ is the coefficient of g in f_x (see (4.1)) and the $\widehat{A}$ matrices are of the form

$$\widehat{A_0}=\alpha_0A_0'\times R_1^{adj} \qquad \widehat{A_{i,b_1,b_2}}=\alpha_{i,b_1,b_2}R_i\times A'_{i,b_1,b_2}\times R_{i+1}^{adj} \qquad \widehat{A_{\ell+1}}=\alpha_{\ell+1}R_{\ell+1}\times A'_{\ell+1}$$

we can expand the $\widehat{A}$ matrices to write $f_x^{(1)}=d_x+\alpha_0\cdot d_x'$, where

$$d_x:=Z_0R_1\left(\prod_{i=1}^{\ell}\alpha_{i,x(i)}A'_{i,x(i)}\right)\alpha_{\ell+1}A'_{\ell+1}\rho_0$$

and d'_x is another polynomial. Here we denote $\rho_0 := \prod_{i=2}^{\ell+1} \det(R_i)$, which arises from the fact that $R_i \times R_i^{adj} = \det(R_i) \cdot I$. Below, we will use the fact that α_0 does not appear in d_x.

Now recall that the A' matrices are constructed as

$$A'_0 := \begin{pmatrix} A_0 & 0^s \end{pmatrix} \qquad A'_{i,b_1,b_2} := \begin{pmatrix} A_{i,b_1,b_2} & \\ & B_{i,b_1,b_2} \end{pmatrix} \qquad A'_{\ell+1} := \begin{pmatrix} A_{\ell+1} \\ B_{\ell+1} \end{pmatrix}$$

where the A matrices are the original branching program input to $\mathcal{O}$. We consider two cases: either

- Q' annihilates $\left\{f_x^{(1)}\right\}_{x\in\mathcal{X}}$ when considered as polynomials in variables Z, R, B, and α (i.e. when only the A matrices are taken to be $\mathbb{Z}_p$-values), or
- it does not, but with probability $\epsilon \geq 1/\mathsf{poly}(\lambda)$ over the distribution on R, B, and α, Q' annihilates the set $\left\{f_x^{(1)}\right\}_{x\in\mathcal{X}}$ when considered as polynomials in variables Z.

Here and throughout the remainder of the proof, we use the phrase "variables Z" to refer to the set of all variables arising from the Z matrices, and similarly for R, B, and α.

We now show that the first case contradicts the (t,s)-BPUA assumption, while the second case is ruled out by the Schwartz-Zippel lemma.

Case 1: Q' annihilates $\left\{f_x^{(1)}\right\}_{x\in\mathcal{X}}$ as polynomials in variables Z, R, B, and α.

Because we can write $f_x^{(1)} = d_x + \alpha_0 \cdot d'_x$, where d_x does not contain the variable α_0, if Q' annihilates $\left\{f_x^{(1)}\right\}_{x\in\mathcal{X}}$ as polynomials in variables Z, R, B, and α, it must also annihilate $\{d_x\}_{x\in\mathcal{X}}$.

Next, we perform the following change of variables: we set each R matrix to be the identity matrix (which in particular induces $\rho_0 = 1$), we set each α scalar to 1, and we set $Z_0 = (uV \quad B_0)$ for a new variable u and new vectors of variables V, B_0 which have lengths w and s respectively (recall that the A and B matrices have dimensions w and s respectively). Applying this change of variables to d_x, we obtain the polynomial $e_x + u \cdot e'_x$, where

$$e_x := B_0 \times \prod_{i\in[\ell]} B_{i,x(i)} \times B_{\ell+1}$$

and e'_x is another polynomial. Because $e_x + u \cdot e'_x$ was obtained from d_x via a change of variables, if Q' annihilates $\{d_x\}_{x\in\mathcal{X}}$ then it must also annihilate $\{e_x + u \cdot e'_x\}_{x\in\mathcal{X}}$. Further, since the variable u does not appear in e_x, Q' must also annihilate $\{e_x\}_{x\in\mathcal{X}}$.

However, this contradicts the (t,s)-BPUA assumption: by construction of inp and ℓ in Section 3, any branching program of the form in Assumption 4.3 can be embedded into the B matrices, and thus there is an efficiently computable distribution on degree-$2^{o(\lambda)}$ polynomials that annihilates all such branching programs with probability $\geq 1/\mathsf{poly}(\lambda)$.

Case 2: $\Pr_{R,B,\alpha}\left[Q' \text{ annihilates } \left\{f_x^{(1)}\right\}_{x\in\mathcal{X}} \text{ as polynomials in variables } Z\right] \geq 1/\mathsf{poly}(\lambda)$.

If Case 1 does not hold, then $Q'(\{f_x^{(1)}\}_{x\in\mathcal{X}})$ must contain some non-zero monomial. View this monomial as being over the variables g and Z, whose coefficient is a non-zero polynomial γ of degree $2^{o(\lambda)}$ in variables R, B, and α. (The degree bound on γ comes from the fact that Q' has degree $2^{o(\lambda)}$ and each $f_x^{(1)}$ has degree $\mathsf{poly}(\lambda)$.)

If Case 2 holds, we must have $\Pr_{R,B,\alpha}[\gamma(R,B,\alpha) = 0] \geq 1/\mathsf{poly}(\lambda)$. However, this contradicts the Schwartz-Zippel lemma, because we are working over the field $\mathbb{Z}_p$ with $p > 2^\lambda$, and the distribution on the variables R, B, α is $2^{-\Omega(\lambda)}$-close to each being uniform and independent. Indeed,

the distributions on the B variables are uniform over $\mathbb{Z}_p$, the distributions on the α variables are uniform over $\mathbb{Z}_p \setminus \{0\}$, and the distributions on the R variables are uniform over $\mathbb{Z}_p$ conditioned on each matrix R_i being non-singular. □

We now prove the lemma that was used in the proof of Theorem 4.5. We will need the following result from [BMSZ16]. Recall that $\overline{f_x}$ denotes the matrix product polynomial that corresponds to evaluating $\widehat{BP}(x)$.

Theorem 4.6 ([BMSZ16])**.** *Fix $x \in \{0,1\}^n$, and consider the following matrices from Section 3: $A'_i := A'_{i,x(i)}$, $\widehat{A}_i := \widehat{A}_{i,x(i)}$, and R_i. Consider also a polynomial f in the entries of the $\widehat{A}$ matrices in which each monomial contains at most one variable from each $\widehat{A}_i$. Let f' be the polynomial derived from f after making the substitution $\widehat{A}_i = R_{i-1}^{adj} \times A'_i \times R_i$, and suppose that f' is identically 0 as a polynomial over the R_i.*

Then either f is identically zero as a polynomial over its formal variables (namely the $\widehat{A}_i$), or else f is a constant multiple of the matrix product polynomial $\overline{f_x} = \widehat{A_0} \times \cdots \times \widehat{A_{\ell+1}}$.

We remark that the proof of this theorem requires that the A' matrices form a non-shortcutting branching program (see Def. 3.1), and that for us this is implied by the distribution on the B matrices and the fact that A is forward non-shortcutting.

Lemma 4.7. *Let BP be any forward-non-shortcutting branching program, and let the model $\mathcal{M}$ from Section 2 be initialized by the obfuscator $\mathcal{O}(BP)$ with parameters t, s as described in Section 3.*

Let $\mathcal{A}$ be an efficient adversary interacting with $\mathcal{M}$, and let $\{h_j\}_{j\in[m]}$ be the set of all handles $\mathcal{A}$ has received that map to a level-$\mathbb{U}$ polynomial with constant term 0 in $\mathcal{M}$'s pre-zero-test table; denote these polynomials by $\{f_j\}_{j\in[m]}$. Assume that $\mathcal{A}$ has not received "WIN" in response to any post-zero-test query.

Then with probability $1-\mathsf{negl}(\lambda)$ over the randomness of $\mathcal{O}$, there exist $\mathsf{poly}(\lambda)$-size sets $\mathcal{X}_1, \ldots, \mathcal{X}_m$ such that: (1) for each $j \in [m]$, f_j is a linear combination of the polynomials $\{\overline{f_x}\}_{x\in\mathcal{X}_j}$, and (2) for each $j \in [m]$ and each $x \in \mathcal{X}_j$, $BP(x) = 0$.

Proof. The proof follows the analysis of [BMSZ16, Thm. 5.1], which builds on [BGK+14]. We assume that the lemma's conclusion holds for $f_1, \ldots, f_{m-1}$, and prove that it holds for f_m with probability $1 - \mathsf{negl}(\lambda)$. This inductively implies the lemma.

As in the proof of Theorem 4.5, let $\overline{f_m}$ be the polynomial over the set of $\mathcal{M}$'s initial elements such that $f_m = \overline{f_m}(\{a_i + g \cdot z_i\})$. Because $\overline{f_m}$ is at level $\mathbb{U}$, we can use the procedure given by [BGK+14, Sec. 6] (cf. [BMSZ16, Lem. 5.3]) to decompose it as

$$\overline{f_m} = \sum_{x\in\mathcal{X}_m} f_{m,x}$$

with equality as formal polynomials, where $\mathcal{X}_m$ is a $\mathsf{poly}(\lambda)$-size set given by the decomposition, and each $f_{m,x}$ is a non-identically-zero polynomial at level $\mathbb{U}$ that only has variables from matrices in $\widehat{BP}$ that correspond to input x.

Notice that f_m has constant term 0 iff $\overline{f_m}(\{a_i\}_i) = 0$. Then following the [BGK+14, Sec. 6] analysis, the independence of the α_{i,b_1,b_2} randomization variables along with the fact that $\overline{f_m}(\{a_i\}_i) = 0$ implies $\Pr[\exists x \in \mathcal{X}_m : f_{m,x}(\{a_i\}_i) \neq 0] < \mathsf{negl}(\lambda)$, where the probability is over $\mathcal{O}$'s randomness. Assume for the remainder that $f_{m,x}(\{a_i\}_i) = 0$ for all $x \in \mathcal{X}_m$, which occurs with probability $1 - \mathsf{negl}(\lambda)$.

Consider the moment just before $\mathcal{A}$ submits the handle h_m (corresponding to f_m) for zero-testing. At this point, since we assume the lemma's conclusion holds for $f_1, \ldots, f_{m-1}$ and that

$\mathcal{A}$ has never received "WIN" in response to any post-zero-test query, $\mathcal{A}$'s view can be completely derived from the set $\{BP(x) \mid x \in \bigcup_{j\in[m-1]} \mathcal{X}_j\}$. In particular, $\mathcal{A}$'s view is independent of the randomness generated by $\mathcal{O}$.

Now fix some $x \in \mathcal{X}_m$. The values $\{a_i\}_i$ are generated by $\mathcal{O}$ from the original branching program BP by choosing the randomization matrices R and the other randomization values α, B, and performing the computation described in Section 3. We can thus view $f_{m,x}$ as a polynomial $f'_{m,x}$ over the R variables whose coefficients are polynomials in the variables α, B. Then because $f_{m,x}$ only has variables from matrices corresponding to input x and is not identically zero, Theorem 4.6 implies that either $f_{m,x}$ is a constant multiple of $\overline{f_x}$, or else $f'_{m,x}$ is not the identically zero polynomial.

Because we assume $f_{m,x}(\{a_i\}_i) = 0$ for the particular sample of $\{a_i\}_i$ generated by $\mathcal{O}$, if $f'_{m,x}$ is not identically zero, then one of two things must have occurred. Either every coefficient of $f'_{m,x}$ became 0 after the choice of α, B, or some choice of α, B yields a fixed $\mathbb{Z}_p$-polynomial that evaluated to 0 on the choice of the R matrices. However, both of these events have probability $1 - \mathsf{negl}(\lambda)$ by the Schwartz-Zippel lemma. Thus, since $\mathcal{A}$'s view (and in particular $f'_{m,x}$) is independent of $\mathcal{O}$'s randomness, we conclude that with probability $1 - \mathsf{negl}(\lambda)$, $f_{m,x}$ is a constant multiple of $\overline{f_x}$.

Finally, note that if $f_{m,x}$ is a (non-zero) constant multiple of $\overline{f_x}$ and $f_{m,x}(\{a_i\}_i) = 0$, then $\overline{f_x}(\{a_i\}_i) = 0$, which is equivalent to $BP(x) = 0$. □

4.2 Further discussion of our assumption

We first note that PRFs such as those in the statement of Theorem 4.4 can be constructed from any boolean NC^1 PRF, provided $s \geq 5\lambda$ and t is a sufficiently large polynomial. This was done explicitly in [GMS16]. The idea is to take λ copies of a width-5, length-t boolean PRF (constructed via [Bar86]), scale the ith copy by 2^i for $i = 0, \ldots, \lambda - 1$, and put them into a block-diagonal BP of width 5λ with appropriate bookend vectors to sum the scaled copies.

We note that for complicated programs whose length is already larger than t, the overhead for protecting against zeroizing attacks is mainly due to increasing the width by s. The multiplicative overhead is thus $(w + s)^2/w^2$ where w is the original width of the branching program. Thus, for many applications, it is likely best to minimize s, potentially at the expense of a slightly larger t. We note that since [GMS16] require $s \geq 5\lambda$, if the original program has width $w = 5$, the multiplicative overhead will be $\approx \lambda^2$, which is quite large. Next, we now describe how to modify [GMS16] to obtain a branching program of *constant* width.

Making the PRF computation have constant width. We now explain that the width s can actually be taken to be a constant. There are many ways to accomplish this. Perhaps the simplest is the following. Ben Or and Cleve [Cle88] show how to convert any arithmetic formula into a matrix branching program consisting of 3×3 matrices, where the matrix product gives

$$\begin{pmatrix} 1 & f(x) & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Then the output $f(x)$ can be selected by multiplying by the appropriate bookend vectors.

For any invertible constant c in the ring, by left- and right- multiplying the branching program by the constant matrices

$$\begin{pmatrix} c & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \text{ and } \begin{pmatrix} c^{-1} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

the product of the branching program matrices becomes

$$\begin{pmatrix} 1 & cf(x) & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Next, by concatenating the branching programs for f_1 and f_2, the result of the matrix product is

$$\begin{pmatrix} 1 & f_1(x) + f_2(x) & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Let $f_0, \ldots, f_{\lambda-1}$ be independent formulas for computing a pseudorandom bit. It is therefore possible to construct a matrix branching program whose matrix product is

$$\begin{pmatrix} 1 & \sum_{i=1}^{\lambda-1} 2^i f_i(x) & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

By multiplying by the appropriate bookend vectors, the result is $\sum_{i=1}^{\lambda-1} 2^i f_i(x)$. By the pseudorandomness of the f_i, this is a pseudorandom value in $[0, 2^\lambda - 1]$.

Varying the assumption strength. We also note that, based on whether we wish t, s to be polynomial, logarithmic, or constant, we can obtain assumptions of varying strength. For example, we can have the following.

Assumption 4.8 (The poly/poly-BPUA assumption)**.** *There exist polynomials t, s such that the (t, s)-BPUA assumption holds.*

Assumption 4.9 (The poly/const-BPUA assumption)**.** *There exists polynomial t and* constant *s such that the (t, s)-BPUA assumption holds.*

Assumption 4.10 (The polylog/const-BPUA assumption)**.** *There exists* polylogarithmic *t and constant s such that the (t, s)-BPUA assumption holds.*

We can thus get a trade-off between efficiency and assumption strength - stronger assumptions (those with smaller s and t) very naturally correspond to more efficient obfuscators.

Dual Input Assumptions. We could have similarly made dual-input versions of the above assumptions. However, we observe that the single input and dual input variants are equivalent, up to constant factors in t and s.

In particular, any single input branching program of length t and width s can be turned into a dual input program of length $t/2$ and width s by pre-multiplying branching program matrices. That is, set $A'_{i,b_0,b_1} = A_{2i-1,b_0} \cdot A_{2i,b_1}$ and $\mathsf{inp}_b(i) = \mathsf{inp}(2i - b)$.

Moreover, any dual input branching program of length t and width s can be converted into a single input branching program of length $2t$ and width $2s$ via the following transformation:

$$A'_{2i-1,b} = \begin{pmatrix} A_{i,b,0} & A_{i,b,1} \\ 0^{s\times s} & 0^{s\times s} \end{pmatrix} \quad A'_{2i,b} = \begin{pmatrix} (1-b)I_s & 0^{s\times s} \\ bI_s & 0^{s\times s} \end{pmatrix} \quad \mathsf{inp}(i) = \begin{cases} \mathsf{inp}_0((i+1)/2) & \text{if } i \text{ is odd} \\ \mathsf{inp}_1(i/2) & \text{if } i \text{ is even} \end{cases}$$

Notice that $A'_{2i-1,b_0} \cdot A'_{2i,b_1} = \begin{pmatrix} A_{i,b_0,b_1} & 0^{s\times s} \\ 0^{s\times s} & 0^{s\times s} \end{pmatrix}$.

References

[AGIS14] Prabhanjan Ananth, Divya Gupta, Yuval Ishai, and Amit Sahai. Optimizing obfuscation: Avoiding Barrington's theorem. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 646–658, 2014.

[AJN+16] Prabhanjan Ananth, Aayush Jain, Moni Naor, Amit Sahai, and Eylon Yogev. Universal constructions and robust combiners for indistinguishability obfuscation and witness encryption. In *Advances in Cryptology - CRYPTO*, 2016.

[App14] Benny Applebaum. Bootstrapping obfuscators via fast pseudorandom functions. In *ASIACRYPT*, 2014.

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

[BGH+15] Zvika Brakerski, Craig Gentry, Shai Halevi, Tancrede Lepoint, Amit Sahai, and Mehdi Tibouchi. Cryptanalysis of the quadratic zero-testing of GGH. Cryptology ePrint Archive, Report 2015/845, 2015. http://eprint.iacr.org/.

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *CRYPTO*, pages 1–18, 2001.

[BGI+12] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2):6, 2012.

[BGJ+15] Nir Bitansky, Shafi Goldwasser, Abhishek Jain, Omer Paneth, Vinod Vaikuntanathan, and Brent Waters. Time-lock puzzles from randomized encodings. Cryptology ePrint Archive, Report 2015/514, 2015. http://eprint.iacr.org/.

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

[BMSZ16] Saikrishna Badrinarayanan, Eric Miles, Amit Sahai, and Mark Zhandry. Post-zeroizing obfuscation: New mathematical tools, and the case of evasive circuits. In *Advances in Cryptology - EUROCRYPT*, pages 764–791, 2016.

[BR14] Zvika Brakerski and Guy N. Rothblum. Virtual black-box obfuscation for all circuits via generic graded encoding. In *11th Theory of Cryptography Conference 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/.

[CGH+15] Jean-Sébastien Coron, Craig Gentry, Shai Halevi, Tancrède Lepoint, Hemanta K. Maji, Eric Miles, Mariana Raykova, Amit Sahai, and Mehdi Tibouchi. Zeroizing without low-level zeroes: New MMAP attacks and their limitations. In *CRYPTO*, 2015.

[CHL+15] Jung Hee Cheon, Kyoohyung Han, Changmin Lee, Hansol Ryu, and Damien Stehlé. Cryptanalysis of the multilinear map over the integers. In *EUROCRYPT*, 2015.

[Cle88] Richard Cleve. Computing algebraic formulas with a constant number of registers.

In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 254–257, New York, NY, USA, 1988. ACM.

[CLR15] Jung Hee Cheon, Changmin Lee, and Hansol Ryu. Cryptanalysis of the new clt multilinear maps. Cryptology ePrint Archive, Report 2015/934, 2015. http://eprint.iacr.org/.

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

[CLT15] Jean-Sebastien Coron, Tancrede Lepoint, and Mehdi Tibouchi. New multilinear maps over the integers. In *CRYPTO*, 2015.

[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.

[GGH15] Craig Gentry, Sergey Gorbunov, and Shai Halevi. Graph-induced multilinear maps from lattices. In *12th Theory of Cryptography Conference (TCC)*, pages 498–527, 2015.

[GMS16] Sanjam Garg, Pratyay Mukherjee, and Akshayaram Srinivasan. Obfuscation without the vulnerabilities of multilinear maps. Cryptology ePrint Archive, Report 2016/390, 2016. http://eprint.iacr.org/.

[Hal15] Shai Halevi. Graded encoding, variations on a scheme. *IACR Cryptology ePrint Archive*, 2015:866, 2015.

[HJ15] Yupu Hu and Huiwen Jia. Cryptanalysis of GGH map. *IACR Cryptology ePrint Archive*, 2015:301, 2015.

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

[MF15] Brice Minaud and Pierre-Alain Fouque. Cryptanalysis of the new multilinear map over the integers. Cryptology ePrint Archive, Report 2015/941, 2015. http://eprint.iacr.org/.

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

[MSZ16] Eric Miles, Amit Sahai, and Mark Zhandry. Annihilation attacks for multilinear maps: Cryptanalysis of indistinguishability obfuscation over GGH13. In *Advances in Cryptology - CRYPTO*, 2016.

[PST14] Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation from semantically-secure multilinear encodings. In *Advances in Cryptology – CRYPTO*, pages 500–517. 2014.

A Straddling set level structure

Here we describe the level structure for the graded encoding scheme that is used by the obfuscator $\mathcal{O}$ when initializing the model $\mathcal{M}$ with the values of $\widehat{BP}$ (see Section 3). This construction is due to Barak et al. [BGK+14], and was used in several subsequent works. It relies on the following notion of a *straddling set system*.[4]

Definition A.1. A *straddling set system* with n entries is a universe set $\mathbb{U}$ and a collection of subsets $\mathbb{S} = \{S_{i,b} \subseteq \mathbb{U}\}_{i\in[n],b\in\{0,1\}}$ such that

1. $\bigcup_{i\in[n]} S_{i,0} = \bigcup_{i\in[n]} S_{i,1} = \mathbb{U}$, and
2. for any distinct $C, D \subseteq \mathbb{S}$ such that $\bigcup_{S\in C} S = \bigcup_{S\in D} S$, there exists $b \in \{0,1\}$ such that $C = \{S_{i,b}\}_{i\in[n]}$ and $D = \{S_{i,1-b}\}_{i\in[n]}$.

For any n, the following is a straddling set system with n entries over the universe $\mathbb{U} = \{1, \ldots, 2n-1\}$ (for a proof see [BGK+14, App. A]).

$$S_{1,0} = \{1\}, S_{2,0} = \{2,3\}, \ldots, S_{i,0} = \{2i-2, 2i-1\}, \ldots, S_{n,0} = \{2n-2, 2n-1\}$$

$$S_{1,1} = \{1,2\}, \ldots, S_{i,1} = \{2i-1, 2i\}, \ldots, S_{n-1,1} = \{2n-3, 2n-2\}, S_{n,1} = \{2n-1\}$$

We now describe the level structure that is used to encode $\widehat{BP}$. For each input index $i \in [n]$, let r_i denote the number of layers in which bit i is read, and create a straddling set system with r_i entries. We denote the universe set of this straddling set system by $\mathbb{U}^{(i)}$, and its subsets by $\{S^{(i)}_{j,b}\}_{j\in[r_i],b\in\{0,1\}}$. The overall universe set is then $\mathbb{U} := \bigcup_{i\in[n]} \mathbb{U}^{(i)} \cup \{L, R\}$, where we assume that the $\mathbb{U}^{(i)}$ are pairwise disjoint, and L and R are new symbols that don't appear in any $\mathbb{U}^{(i)}$.

Then, for each matrix $\widehat{A_{j,b_1,b_2}}$ in $\widehat{BP}$, each entry of this matrix is encoded at level

$$S^{(\mathsf{inp}(j)_1)}_{k_1,b_1} \cup S^{(\mathsf{inp}(j)_2)}_{k_2,b_2}$$

where k_1, k_2 are defined such that layer j is the k_1-th layer in which input bit $\mathsf{inp}(j)_1$ is read and the k_2-th layer in which input bit $\mathsf{inp}(j)_2$ is read. Finally, each entry of $\widehat{A_0}$ is encoded at level $\{L\}$, and each entry of $\widehat{A_{\ell+1}}$ is encoded at level $\{R\}$.

[4] For the analysis that we borrow from [BGK+14, BMSZ16], namely Lemma 4.7, we will not need the *strong* straddling set systems due to [MSW14].