Protecting Obfuscation against Algebraic Attacks

Boaz Barak[1], Sanjam Garg[2,⋆], Yael Tauman Kalai[1], Omer Paneth[3,⋆⋆], and Amit Sahai[4,⋆⋆⋆]

[1] Microsoft Research
[2] IBM Research
[3] Boston University
[4] UCLA

Abstract. Recently, Garg, Gentry, Halevi, Raykova, Sahai, and Waters (FOCS 2013) constructed a general-purpose obfuscating compiler for $\mathbf{NC}^1$ circuits. We describe a simplified variant of this compiler, and prove that it is a virtual black box obfuscator in a generic multilinear map model. This improves on Brakerski and Rothblum (eprint 2013) who gave such a result under a strengthening of the Exponential Time Hypothesis. We remove this assumption, and thus resolve an open question of Garg *et al.* As shown by Garg *et al.*, a compiler for $\mathbf{NC}^1$ circuits can be bootstrapped to a compiler for all polynomial-sized circuits under the learning with errors (LWE) hardness assumption.

Our result shows that there is a candidate obfuscator that cannot be broken by algebraic attacks, hence reducing the task of creating secure obfuscators in the plain model to obtaining sufficiently strong security guarantees on candidate instantiations of multilinear maps.

1 Introduction

The goal of general-purpose program obfuscation is to make an arbitrary computer program "unintelligible" while preserving its functionality. At least as

⋆ Research conducted while at the IBM Research, T.J.Watson funded by NSF Grant No.1017660.

⋆⋆ Work done while the author was an intern at Microsoft Research New England. Supported by the Simons award for graduate students in theoretical computer science and an NSF Algorithmic foundations grant 1218461.

⋆⋆⋆ Work done in part while visiting Microsoft Research, New England. Research supported in part from a DARPA/ONR PROCEED award, 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 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.

P.Q. Nguyen and E. Oswald (Eds.): EUROCRYPT 2014, LNCS 8441, pp. 221–238, 2014.

far back as the work of Diffie and Hellman in 1976 [7][1], researchers have contemplated applications of general-purpose obfuscation. The first mathematical definitions of obfuscation were given by Hada [11] and Barak, Goldreich, Impagliazzo, Rudich, Sahai, Vadhan, and Yang [2].[2] Barak et al. also enumerated several additional applications of general-purpose obfuscation, ranging from software intellectual property protection and removing random oracles, to eliminating software watermarks. However, until 2013, even heuristic constructions for general-purpose obfuscation were not known.

This changed with the work of Garg, Gentry, Halevi, Raykova, Sahai, and Waters in 2013 [9], which gave the first candidate construction for a general-purpose obfuscator. At the heart of their construction is an obfuscator for log-depth ($\mathbf{NC}^1$) circuits, building upon a simplified subset of the Approximate Multilinear Maps framework of Garg, Gentry, and Halevi [8] that they call Multilinear Jigsaw Puzzles. They proved that their construction achieves a notion called indistinguishability obfuscation (see below for further explanation), under a complex new intractability assumption. They then used fully homomorphic encryption to bootstrap this construction to work for all circuits, proving their transformation secure under the Learning with Error (LWE) assumption, a well-studied intractability assumption.

Our result— protecting against algebraic attacks. Given the importance of general-purpose obfuscation, it is imperative that we gain as much confidence as possible in candidates for general-purpose obfuscation. Potential attacks on the [9] obfuscator can be classified into two types— attacks on the underlying Multilinear Jigsaw Puzzle construction, and attacks on the obfuscation construction that treat the Multilinear Jigsaw Puzzle as an ideal black box. [8] gave some cryptanalytic evidence for the security of their Approximate Multilinear Maps candidate (this evidence immediately extends to Mathematical Jigsaw Puzzles, since it is a weaker primitive), and there is also an alternative candidate [6] for such maps. Our focus in this paper is to find out whether there exists a *purely algebraic* attack against the candidate obfuscation schemes, or whether any attack against the scheme must rely on some weakness of the underlying Multilinear Jigsaw Puzzle (i.e., some deviation of the implementation from the ideal model). Indeed, [9] pose the problem of proving that there exist no generic multilinear attacks against their core $\mathbf{NC}^1$ scheme as a major open problem in their work.[3]

[1] Diffie and Hellman suggested the use of general-purpose obfuscation to convert private-key cryptosystems to public-key cryptosystems.

[2] The work of [2] is best known for their constructions of "unobfuscatable" classes of functions $\{f_s\}$ that roughly have the property that given *any* circuit evaluating f_s, one can extract the secret s, yet given only *black-box* access to f_s, the secret s is hidden. We will discuss the implications of this for our setting below.

[3] [9] did rule out a certain subset of algebraic attacks which fall under a model they called the "generic colored matrix model". However, this model assumes that an adversary can only attack the schemes by performing a limited subset of matrix operations, and does not prove any security against an adversary that can perform algebraic operations on the individual entries of the matrices.

This problem was first addressed in the recent work of Brakerski and Rothblum [4], who constructed a variant of the [9] candidate obfuscator, and proved that it is an indistinguishability obfuscation against all generic multilinear attacks. They also proved that their obfuscator achieves the strongest definition of security for general-purpose obfuscation — Virtual Black Box (VBB) security — against all generic multilinear attacks, albeit under an unproven assumption they introduce as the Bounded Speedup Hypothesis, which strengthens the Exponential Time Hypothesis from computational complexity.[4]

In this work, we resolve the open problem of [9] completely, by removing the need for this additional assumption. More specifically, we describe a different (and arguably simpler) variant of the construction of [9], for which we can prove that it achieves Virtual Black Box security against all generic multilinear attacks, *with no further assumptions*. Our result gives evidence for the soundness of [9]'s approach for building obfuscators based on Multilinear Jigsaw Puzzles.

Notions of Security and attacks. In this work, we focus on arguing security against a large class of natural algebraic attacks, captured in the *generic multilinear* model. Intuitively speaking, the generic multilinear model imagines an exponential-size collection of "groups" $\{G_S\}$, where the subscript S denotes a subset $S \subseteq \{1, 2, \ldots, k\}$. Each of these groups is a separate copy of $\mathbb{Z}_p$, under addition, for some fixed large random prime p. The adversary is initially given some collection of elements from various groups. However, the only way that the adversary can process elements of these groups is through access to an oracle $\mathcal{M}$ that performs the following three operations[5]:

- **Addition:** $G_S \times G_S \to G_S$, defined in the natural way over $\mathbb{Z}_p$, for all $S \subset \{1, 2, \ldots, k\}$.
- **Negation:** $G_S \to G_S$, defined in the natural way over $\mathbb{Z}_p$, for all $S \subset \{1, 2, \ldots, k\}$.
- **Multiplication:** $G_S \times G_T \to G_{S \cup T}$, defined in the natural way over $\mathbb{Z}_p$, for all $S, T \subset \{1, 2, \ldots, k\}$, where $S \cap T = \emptyset$. Note that the constraint that $S \cap T = \emptyset$ intuitively captures why we call this a *multilinear* model.

These operations capture precisely the algebraic operations supported by the Multilinear Jigsaw Puzzles of [9].

With the algebraic attack model defined, the next step is to consider what security property we would like to achieve with respect to this attack model. We first recall two security notions for obfuscation – indistinguishability obfuscation

[4] Roughly speaking, the Bounded Speedup Hyptothesis says that there is some $\epsilon > 0$ such that for every subset $\mathcal{X}$ of $\{0,1\}^n$, any circuit C that solves SAT on all inputs in $\mathcal{X}$ must have size at least $|\mathcal{X}|^\epsilon$. The Exponential Time Hypothesis is recovered by considering $\mathcal{X} = \{0,1\}^n$. The exponent of the polynomial slowdown of the [4] simulator is a function of ϵ.

[5] In the technical exposition, we discuss how it is enforced that the adversary can *only* access the elements of the group via the oracles. For this intuitive exposition, we ask the reader to simply imagine that an algebraic adversary is defined to be limited in this way.

(iO) security and Virtual Black-Box (VBB) security – and state them both in comparable language, in the generic multilinear model. Below, we write "generic adversary" or "generic distinguisher" to refer to an algorithm that has access to the oracle $\mathcal{M}$ described above.

Indistinguishability obfuscation[6] requires that for every polynomial-time generic adversary, there exists an *computationally unbounded* simulator, such that for every circuit C, no polynomial-time generic distinguisher can distinguish the output of the adversary given the obfuscation of C as input, from the output of the simulator given oracle access to C, where the simulator can make an *unbounded* number of queries to C. Virtual Black-Box obfuscation[7] requires that for every polynomial-time generic adversary, there exists a *polynomial-time* simulator, such that for every circuit C, no polynomial-time generic distinguisher can distinguish the output of the adversary given the obfuscation of C as input, from the output of the simulator given oracle access to C, where the simulator can make a *polynomial* number of queries to C.

In our work, we focus on proving the Virtual Black-Box definition of security against generic attacks. We do so for several reasons:

- Our first, and most basic, reason is that Virtual Black-Box security is the strongest security notion of obfuscation we are aware of, and so proving VBB security against generic multilinear attacks is, mathematically speaking, the strongest result we could hope to prove. As we can see from the definitions above, the definition of security provided by the VBB definition is significantly stronger than the indistinguishability obfuscation definition. As such, it represents the natural end-goal for research on proving resilience to such algebraic attacks.
  This may seem surprising in light of the negative results of [2], who showed that there exist (contrived) families of "unobfuscatable" functions for which the VBB definition is impossible to achieve *in the plain model*. However, we stress that this result does not apply to security against generic multilinear attacks. Thus it does not present a barrier to the goal of proving VBB security against generic multilinear attacks.
- Given the existence of "unobfuscatable" function families, how can we interpret a result showing VBB security against generic attacks, in terms of the real-world applicability of obfuscation? One plausible interpretation is that it offers heuristic evidence that our obfuscation mechanism will offer strong security for "natural" functions, that do not have the self-referential properties of the [2] counter-examples. This is similar to the heuristic evidence

[6] The formulation of indistinguishability obfuscation sketched here was used, for example, in [9].

[7] We note that we are referring to a stronger definition of VBB obfuscation than the one given in [2], which limits the adversary to only outputting one bit. In our definition, the adversary can output arbitrary length strings. This stronger formulation of VBB security implies all other known meaningful security definitions for obfuscation, including natural definitions that are not known to be implied by the one-bit-output formulation of VBB security.

given by a proof in the Random Oracle Model. We stress, however, that our result cannot offer any specific theoretical guidance on *which* function families can be VBB-obfuscated in the plain model, and which cannot.

– Finally, our VBB result against generic attacks suggests that there is a significant gap between what security is *actually* achieved by our candidate in the plain model, and the best security *definitions* for obfuscation that we have in the plain model. This suggests a research program for studying relaxations of VBB obfuscation that could plausibly be achievable in the plain model. Indistinguishability Obfuscation is one such example, but other notions have been suggested in the literature, and it's quite possible we haven't yet found the "right" notion. For every such definition of obfuscation X, one can of course make the assumption that our candidate is "X secure" in the plain model, but in fact our VBB proof in the generic multilinear model shows that "X security" of our candidate will follow from a concrete intractability assumption on the Multilinear Jigsaw Puzzle implementation *that is unrelated to our specific obfuscation candidate* (see below for more details).

*Remark 1.1 (*Capturing a Generic Model by Meta-Assumptions*).* While a generic model allows us to precisely define and argue about large classes of algebraic attacks, it is unsatisfying because any such oracle model, by definition, cannot be achieved in the plain model. Thus, we would like to capture as much as we can of a generic model by means of what we would call a "Meta-Assumption." Intuitively, a Meta-Assumption specifies conditions under which the only attacks that are possible in the plain model with a specific instantiation of the oracle, are those that are possible in the oracle model itself – where the conditions that the Meta-Assumption imposes allow the assumption to be plausible. For example, one can consider the Decisional Diffie Hellman (DDH) assumption as a meta assumption on the instantiation of the group $\mathbb{Z}_q$ as a multiplicative subgroup of $\mathbb{Z}^*_{p=kq+1}$, stipulating that certain attacks that would be infeasible in the ideal setting, are also infeasible when working with the actual encoding of the group elements.

1.1 Our Techniques

The starting point for our construction is a simplified form of the construction of [9]. That work used the fact that one can express an $\mathbf{NC}^1$ computation as a *Branching Program*, which is a sequence of $2n$ permutations (or more generally, functions) $\{B_{i,\sigma}\}_{i\in[n],\sigma\in\{0,1\}}$. The program is evaluated on an input $x \in \{0,1\}^\ell$ by applying for $i = 1, \ldots, n$ the permutation $B_{i,x_{\mathsf{inp}(i)}}$ where inp is some map from $[n]$ to $[\ell]$ that says which input bit the branching program looks at the i^{th} step. The output of the program is obtained based on the composition of all these permutations; that is, we have some permutation P_{accept} (without loss of generality, the identity) and say that the output is 1 if the composition is equal to P_{accept} and the output is 0 otherwise.[8] We can identify these permutations with matrices,

[8] Barrington's Theorem [3] shows that these permutations can be taken to have a finite domain (in fact, 5) but for our construction, a domain of $\mathsf{poly}(\ell)$ size is fine.

and so evaluating the program amounts to matrix multiplication. Matrix multiplication is an algebraic (and in fact multilinear) operation, that can be done in a group supporting multilinear maps. Thus a naive first attempt at obfuscation of an $\mathbf{NC}^1$ computation would be to encode all the elements of the matrices $\{B_{i,\sigma}\}_{i\in[n],\sigma\in\{0,1\}}$ in the multilinear maps setting (using disjoint subsets to encode elements of matrices that would be multiplied together, e.g., by encoding the elements of $B_{i,\sigma}$ in the group $G_{\{i\}}$). This would allow to run the computation on every $x \in \{0,1\}^\ell$. However, as an obfuscation it would be completely insecure, since it will also allow an adversary to perform tricks such as "mixing inputs" by starting the computation on a particular input x and then at some step switching to a different input x'. Even if it fixes some particular input $x \in \{0,1\}^\ell$, the adversary might learn not just the product of the n matrices $B_{1,x_{\mathsf{inp}(1)}}, \ldots, B_{n,x_{\mathsf{inp}(n)}}$ but also information about partial products. To protect against this latter attack, [9] used a trick of Kilian [12] where instead of the matrices $\{B_{i,\sigma}\}_{i\in[n],\sigma\in\{0,1\}}$ they published the matrices $\{B'_{i,\sigma} = R_{i-1}^{-1} B_{i,\sigma} R_i\}_{i\in[n],\sigma\in\{0,1\}}$ where R_0, R_n are the identity and $R_1, \ldots, R_{n-1}$ are random permutation matrices.[9] We follow the same approach. The crucial obstacle is that in our setting, because we need to supply a single program that works on *all* inputs $x \in \{0,1\}^\ell$, we need to reveal both the matrix $B_{i,0}$ and the matrix $B_{i,1}$, and will need to multiply them both with the same random matrix. Unfortunately, Kilian's trick does not guarantee security in such a setting. It also does not protect against the "mixed input" attack described above.

We deviate from the works [9, 4] in the way we handle the above issues. Specifically, the most important difference is that we employ specially designed set systems in our use of the generic multilinear model. Roughly speaking, in the original work of [9], the encoding of the elements of matrix $B'_{i,\sigma}$ was in the group $G_{\{i\}}$. In contrast, in our obfuscation, while the actual elements from $\mathbb{Z}_p$ that we use are very similar to those used in [9], these elements will live in groups G_S where the sets S will come from specially designed set systems. To illustrate this idea, consider the toy example where $\ell = 1$ and $n = 2$. That is, we have a single input bit $x \in \{0,1\}$ and 4 matrices $B'_{1,0}, B'_{1,1}, B'_{2,0}, B'_{2,1}$. We want to supply encodings that will allow computing the products $B'_{1,0}B'_{2,0}$ and $B'_{1,1}B'_{2,1}$, but not any of the "mixed products" such as $B'_{1,0}B'_{2,1}$ which corresponds to pretending the input bit is equal to 0 in the first step of the branching program, and equal to 1 in the second step. The idea is that our groups will be of the form $\{G_S\}$ where S is a subset of the universe $\{1,2,3\}$. We will encode the elements of $B_{1,0}$ in $G_{\{1,2\}}$, the elements of $B_{1,1}$ in $G_{\{1\}}$, the elements of $B_{2,0}$ in $G_{\{3\}}$, and the elements of $B_{2,1}$ in $G_{\{2,3\}}$. One can see that one can use our oracle to obtain an encoding of the two matrices corresponding to the "proper" products in $G_{\{1,2,3\}}$, but it is not possible to compute the "mixed product" since it would involved multiplying elements in G_S and G_T for non-disjoint S and T. This idea

[9] Instead of using R_0, R_{n+1} as the identity, [9] and us added some additional encoding of elements they called "bookends". We ignore this detail in this section's high level description. We also defer discussion of an additional trick of multiplying each element in $B'_{i,\sigma}$ by a scalar $\alpha_{i,\sigma}$.

can be easily extended to the case of larger ℓ and n, and can be used to rule out the mixed product attack.

However, the idea above still does not rule out "partial evaluation attacks", where the adversary might try to learn, for example, whether the first k steps of the branching program evaluate to the same permutation regardless of the value of the first bit of x. To do that we enhance our set system by creating interlocking sets that combine several copies of the straddling set systems above. Roughly speaking, these interlocking sets ensure that the adversary cannot create "interesting" combinations of the encoded elements, without in effect committing to a particular input $x \in \{0,1\}^\ell$. This prevents the adversary from creating polynomials that combine terms corresponding to a super-polynomial set of different inputs. In contrast, in the recent work of [4], this was accomplished by means of a reduction to the Bounded Speedup Hypothesis. In contrast, our generic proof does not use any assumptions except the properties of our set systems.

The second deviation in our construction from that of [9] is in our usage of the random scalar values $\{\alpha_{i,\sigma}\}_{i\in[n],\sigma\in\{0,1\}}$ that are used to multiply every element in the encoding of $B'_{i,\sigma}$. In [9] these random scalars $\alpha_{i,b}$ were used for two purposes: First, they were chosen with specific multiplicative constraints in order to prevent "input mixing" attacks as described above (a similar multiplicative bundling method was used by [4] as well). As noted above, we no longer need this use of the $\alpha_{i,b}$ values as this is handled by our set systems. The second purpose these values served was to provide a "per-input" randomization in polynomial terms created by the adversary. We continue the use of this role of the $\alpha_{i,b}$ values, leveraging this "per-input" randomization using a method of explicitly invoking Kilian's randomization technique. This is similar to (but arguably simpler than) the beautiful use of Kilian's randomization technique in the recent work of [4].

Additional Related Work. Our work deals with analyzing candidate general-purpose obfuscators in an idealized mathematical model (the generic multilinear model). There has also been recent work suggesting general-purpose obfuscators in idealized mathematical models which currently do not have candidate instantiations in the standard model: the work of [5] describes a general-purpose obfuscator for NC^1 in a generic group setting with a group $G = G_1 \times G_2 \times G_3 \times G_4$, where G_1 is a pseudo-free Abelian group, G_2 and G_3 are pseudo-free non-Abelian groups, and G_4 is a group supporting Barrington's theorem, such as S_5. In this generic setting, obfuscator described by [5] achieves Virtual Black-Box security. However, no candidate methods for heuristically implementing such a group G are known, and therefore, the work of [5] does not describe a candidate general-purpose obfuscator at this time, though this may change with future work[10].

We note that question of whether there exists any oracle with respect to which virtual black-box obfuscation for general circuits is possible is a trivial question: one can consider a universal oracle that (1) provides secure encryptions e_C for any circuit C to be obfuscated, and (2) given an encrypted circuit

[10] Indeed, one way to obtain a heuristic generic group G is by building it using a general-purpose obfuscator, but this would not be useful for the work of [5], since their goal is a general-purpose obfuscator.

e_C and an input x outputs $C(x)$. The only way we can see this "solution" as being interesting is if one considers implementing this oracle with trusted hardware. The work of Goyal *et al.* [10] shows that there exists an oracle that can be implemented with trusted hardware of size that is only a fixed polynomial in the security parameter, with respect to which virtual black-box obfuscation is possible. However, once again, the focus of our paper is to consider oracles that abstract the natural algebraic functionality underlying actual plain-model candidates for general-purpose obfuscation.

2 Preliminaries

In this section we define the notion of "virtual black-box" obfuscation in an idealized model, we recall the definition of branching programs and describe a "*dual-input*" variant of branching programs used in our construction.

2.1 "Virtual Black-Box" Obfuscation in an Idealized Model

Let $\mathcal{M}$ be some oracle. We define obfuscation in the $\mathcal{M}$-idealized model. In this model, both the obfuscator and the evaluator have access to the oracle $\mathcal{M}$. However, the function family that is being obfuscated does not have access to $\mathcal{M}$

Definition 2.1 ("Virtual Black-Box" Obfuscation in an $\mathcal{M}$-idealized model). *For a (possibly randomized) oracle $\mathcal{M}$, and a circuit class $\{\mathcal{C}_\ell\}_{\ell\in\mathbb{N}}$, 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, 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 $\mathcal{M}$:*

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

  where the probability is over the coins of $\mathcal{O}$.
- *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 $\mathcal{S}$, and a negligible function μ such that for all PPT distinguishers D, for every $\ell \in \mathbb{N}$ and every $C \in \mathcal{C}_\ell$:*

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

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

Remark 2.1. We note that the definition above is stronger than the definition of VBB obfuscation given in [2], in that it allows adversaries to output an unbounded number of bits.

Definition 2.2 ("Virtual Black-Box" Obfuscation for $\mathbf{NC}^1$ in an $\mathcal{M}$-idealized model). *We say that $\mathcal{O}$ is a "Virtual Black-Box" Obfuscator for $\mathbf{NC}^1$ in the $\mathcal{M}$-idealized model, if for every circuit class $\mathcal{C} = \{\mathcal{C}_\ell\}_{\ell\in\mathbb{N}}$ such that every circuit in $\mathcal{C}_\ell$ is of size $\mathsf{poly}(\ell)$ and of depth $O(\log(\ell))$, $\mathcal{O}$ is a "Virtual Black-Box" Obfuscator for $\mathcal{C}$ in the $\mathcal{M}$-idealized model.*

2.2 Branching Programs

The focus of this paper is on obfuscating *branching programs*, which are known to be powerful enough to simulate $\mathbf{NC}^1$ circuits.

A branching program consists of a sequence of steps, where each step is defined by a pair of permutations. In each step the the program examines one input bit, and depending on its value the program chooses one of the permutations. The program outputs 1 if and only if the multiplications of the permutations chosen in all steps is the identity permutation.

Definition 2.3 (Oblivious Matrix Branching Program). *A* branching program *of width w and length n for ℓ-bit inputs is given by a permutation matrix $P_{\mathsf{reject}} \in \{0,1\}^{w\times w}$ such that $P_{\mathsf{reject}} \neq I_{w\times w}$, and by a sequence:*

$$\mathsf{BP} = \left(\mathsf{inp}(i), B_{i,0}, B_{i,1}\right)_{i=1}^{n} ,$$

where each $B_{i,b}$ is a permutation matrix in $\{0,1\}^{w\times w}$, and $\mathsf{inp}(i) \in [\ell]$ is the input bit position examined in step i. The output of the branching program on input $x \in \{0,1\}^\ell$ is as follows:

$$BP(x) \stackrel{def}{=} \begin{cases} 1 & \text{if } \prod_{i=1}^{n} B_{i,x_{\mathsf{inp}(i)}} = I_{w\times w} \\ 0 & \text{if } \prod_{i=1}^{n} B_{i,x_{\mathsf{inp}(i)}} = P_{\mathsf{reject}} \\ \perp & \text{otherwise} \end{cases}$$

The branching program is said to be oblivious *if $\mathsf{inp} : [n] \to [\ell]$ is a fixed function, independent of the function being evaluated.*

Theorem 2.1 ([3]). *For any depth-d fan-in-2 boolean circuit C, there exists an oblivious branching program of width 5 and length at most 4^d that computes the same function as the circuit C.*

Remark 2.2. In our obfuscation construction we do not require that the branching program is of constant width. In particular we can use any reductions that result in a polynomial size branching program.

In our construction we will obfuscate a variant of branching programs that we call *dual-input* branching programs. Instead of reading one input bit in every step, a dual-input branching program inspects a pair of input bits and chooses a permutation based on the values of both bits.

Definition 2.4 (Dual-Input Branching Program) . *A* Oblivious dual-input *branching program of width w and length n for ℓ-bit inputs is given by a permutation matrix $P_{\mathsf{reject}} \in \{0,1\}^{w \times w}$ such that $P_{\mathsf{reject}} \neq I_{w \times w}$, and by a sequence*

$$\mathsf{BP} = \left(\mathsf{inp}_1(i), \mathsf{inp}_2(i), \{B_{i,b_1,b_2}\}_{b_1,b_2 \in \{0,1\}}\right)_{i=1}^{n},$$

where each B_{i,b_1,b_2} is a permutation matrix in $\{0,1\}^{w \times w}$, and $\mathsf{inp}_1(i), \mathsf{inp}_2(i) \in [\ell]$ are the positions of the input bits inspected in step i. The output of the branching program on input $x \in \{0,1\}^{\ell}$ is as follows:

$$\mathsf{BP}(x) \stackrel{def}{=} \begin{cases} 1 & \text{if } \prod_{i=1}^{n} B_{i,x_{\mathsf{inp}_1(i)},x_{\mathsf{inp}_2(i)}} = I_{w \times w} \\ 0 & \text{if } \prod_{i=1}^{n} B_{i,x_{\mathsf{inp}_1(i)},x_{\mathsf{inp}_2(i)}} = P_{\mathsf{reject}} \\ \perp & \text{otherwise} \end{cases}$$

As before, the dual-input branching program is said to be oblivious *if both $\mathsf{inp}_1 : [n] \to [\ell]$ and $\mathsf{inp}_2 : [n] \to [\ell]$ are fixed functions, independent of the function being evaluated.*

Note that any branching program can be simulated by a dual-input branching program with the same width and length, since the dual-input branching program can always "ignore" one input bit in each pair. Moreover, note that any dual-input branching program can be simulated by a branching program with the same width and with length that is twice the length of the dual-input branching program.

3 Straddling Set System

In this section, we define the notion of a *straddling set system*, and prove combinatorial properties regarding this set system. This set system will be an ingredient in our construction, and the combinatorial properties that we establish will be used in our generic proof of security.

Definition 3.1. *A* straddling set system *with n entries is a collection of sets $\mathbb{S}_n = \{S_{i,b}, : i \in [n], b \in \{0,1\}\}$ over a universe U, such that*

$$\cup_{i \in [n]} S_{i,0} = \cup_{i \in [n]} S_{i,1} = U$$

and for every distinct non-empty sets $C, D \subseteq \mathbb{S}_n$ we have that if:

1. *(Disjoint Sets:) C contains only disjoint sets. D contains only disjoint sets.*
2. *(Collision:) $\cup_{S \in C} S = \cup_{S \in D} S$*

Then, it must be that $\exists\ b \in \{0,1\}$:

$$C = \{S_{j,b}\}_{j \in [n]} \ , \ D = \{S_{j,(1-b)}\}_{j \in [n]} \ .$$

Therefore, in a straddling set system, the only exact covers of the universe U are $\{S_{j,0}\}_{j \in [n]}$ and $\{S_{j,1}\}_{j \in [n]}$.

Construction 3.1. *Let* $\mathbb{S}_n = \{S_{i,b},: i \in [n], b \in \{0,1\}\}$*, over the universe* $U = \{1, 2, \ldots, 2n-1\}$*, where:*

$S_{1,0} = \{1\}$, $S_{2,0} = \{2,3\}$, $S_{3,0} = \{4,5\}$, $\ldots$, $S_{i,0} = \{2i-2, 2i-1\}$, $\ldots$, $S_{n,0} = \{2n-2, 2n-1\}$*; and,*

$S_{1,1} = \{1,2\}$, $S_{2,1} = \{3,4\}$, $\ldots$, $S_{i,1} = \{2i-1, 2i\}$, $\ldots$, $S_{n-1,1} = \{2n-3, 2n-2\}$, $S_{n,1} = \{2n-1\}$.

The proof that Construction 3.1 satisfies the definition of a straddling set system is straightforward and is given in the full version of this work [1].

4 The Ideal Graded Encoding Model

In this section describe the ideal graded encoding model where 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 [8]. Roughly, $\mathcal{M}$ will maintain a list of *elements* and will allow a user to perform valid arithmetic operations over these elements. We start by defining the an algebra over elements.

Definition 4.1. *Given a ring* R *and a universe set* U*, an element is a pair* (α, S) *where* $\alpha \in R$ *is the* value *of the element and* $S \subseteq U$ *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* $(\alpha(e_1) \cdot \alpha(e_2), S(e_1) \cup S(e_2))$.

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 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 element in L. After $\mathcal{M}$ has been initialize, all subsequent calls to the initialization interfaces 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 $S(e) \neq U$ the call fails. Otherwise, $\mathcal{M}$ returns 1 if $\alpha(e) = 0$, and returns 0 if $\alpha(e) \neq 0$.

5 Obfuscation in the Ideal Graded Encoding Model

In this section we describe our "virtual black-box" obfuscator $\mathcal{O}$ for $\mathbf{NC}^1$ in the ideal graded encoding model.

Input. The obfuscator $\mathcal{O}$ takes as input a circuit and transforms it into an oblivious dual-input branching program BP of width w and length n for ℓ-bit inputs:

$$\mathsf{BP} = \left(\mathsf{inp}_1(i), \mathsf{inp}_2(i), \{B_{i,b_1,b_2}\}_{b_1,b_2 \in \{0,1\}}\right)_{i=1}^{n}.$$

Recall that each B_{i,b_1,b_2} is a permutation matrix in $\{0,1\}^{w \times w}$, and $\mathsf{inp}_1(i)$, $\mathsf{inp}_2(i) \in [\ell]$ are the positions of the input bits inspected in step i. Without loss of generality, we make the following assumptions on the structure of the brunching program BP:

- In every step BP inspects two different input bits; that is, for every step $i \in [n]$, we have $\mathsf{inp}_1(i) \neq \mathsf{inp}_2(i)$.
- Every pair of different input bits are inspected in some step of BP; that is, for every $j_1, j_2 \in [\ell]$ such that $j_1 \neq j_2$ there exists a step $i \in [n]$ such that $(\mathsf{inp}_1(i), \mathsf{inp}_2(i)) = (j_1, j_2)$.
- Every bit of the input is inspected by BP exactly ℓ' times. More precisely, for input bit $j \in [\ell]$, we denote by $\mathsf{ind}(j)$ the set of steps that inspect the j'th bit:

  $$\mathsf{ind}(j) = \{i \in [n] : \mathsf{inp}_1(i) = j\} \cup \{i \in [n] : \mathsf{inp}_2(i) = j\} \quad .$$

  We assume that for every input bit $j \in [\ell]$, $|\mathsf{ind}(j)| = \ell'$. Note that in every step, the j'th input bit can be inspected at most once.

Randomizing. Next, the Obfuscator $\mathcal{O}$ "randomizes" the branching program BP as follows. First, $\mathcal{O}$ samples a prime p of length $\Theta(n)$. Then, $\mathcal{O}$ samples random and independent elements as follows:

- Non-zero scalars $\{\alpha_{i,b_1,b_2} \in \mathbb{Z}_p : i \in [n], b_1, b_2 \in \{0,1\}\}$.
- Pair of vectors $\mathbf{s}, \mathbf{t} \in \mathbb{Z}_p^w$.
- $n+1$ random full-rank matrices $R_0, R_1, \ldots, R_n \in \mathbb{Z}_p^{w \times w}$.

Finally, $\mathcal{O}$ computes the pair of vectors:

$$\tilde{\mathbf{s}} = \mathbf{s}^t \cdot R_0^{-1}, \quad \tilde{\mathbf{t}} = R_n \cdot \mathbf{t} \ ,$$

and for every $i \in [n]$ and $b_1, b_2 \in \{0,1\}$, $\mathcal{O}$ computes the matrix:

$$\tilde{B}_{i,b_1,b_2} = R_{i-1} \cdot B_{i,b_1,b_2} \cdot R_i^{-1}.$$

Initialization. For every $j \in [\ell]$, let $\mathbb{S}^j$ be a straddling set system with ℓ' entries over a set U_j, such that the sets $U_1, \ldots, U_\ell$ are disjoint. Let $U = \bigcup_{j \in [\ell]} U_j$, and let B_s and B_t be sets such that U, B_s, B_t are disjoint. We associate the set system $\mathbb{S}^j$ with the j'th input bit. We index the elements of $\mathbb{S}^j$ by the steps of the branching program BP that inspect the j'th input. Namely,

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

For every step $i \in [n]$ and bits $b_1, b_2 \in \{0,1\}$ we denote by $S(i, b_1, b_2)$ the union of pairs of sets that are indexed by i:

$$S(i, b_1, b_2) = S^{\mathsf{inp}_1(i)}_{i,b_1} \cup S^{\mathsf{inp}_2(i)}_{i,b_2} \ .$$

Note that by the way we defined the set $\mathsf{ind}(j)$ for input bit $j \in [\ell]$, and by the way the elements of $\mathbb{S}^j$ are indexed, indeed, $S^{\mathsf{inp}_1(i)}_{i,b_1} \in \mathbb{S}^{\mathsf{inp}_1(i)}$ and $S^{\mathsf{inp}_2(i)}_{i,b_2} \in \mathbb{S}^{\mathsf{inp}_2(i)}$.

$\mathcal{O}$ initializes the oracle $\mathcal{M}$ with the ring $\mathbb{Z}_p$, the universe set $U \cup B_s \cup B_t$ and with the following initial elements:

$$\begin{aligned}
&(\mathbf{s} \cdot \mathbf{t}, B_s \cup B_t), \\
&\left\{(\tilde{\mathbf{s}}[j], B_s), (\tilde{\mathbf{t}}[j], B_t)\right\}_{j \in [w]} \\
&\{(\alpha_{i,b_1,b_2}, S(i,b_1,b_2))\}_{i \in [n], b_1, b_2 \in \{0,1\}} \\
&\left\{(\alpha_{i,b_1,b_2} \cdot \tilde{B}_{i,b_1,b_2}[j,k], S(i,b_1,b_2))\right\}_{i \in [n], b_1, b_2 \in \{0,1\}, j,k \in [w]}
\end{aligned}$$

$\mathcal{O}$ receives back a list of handles. We denote the handle to the element (α, S) by $[\alpha]_S$. For a matrix M, $[M]_S$ denotes a matrix of handles such that $[M]_S[j,k]$ is the handle to the element $(M[j,k], S)$. Using this notation, $\mathcal{O}$ receives back the following handles:

$$\begin{aligned}
&\left[\tilde{\mathbf{s}}\right]_{B_s}, \quad \left[\tilde{\mathbf{t}}\right]_{B_t}, \quad [\mathbf{s} \cdot \mathbf{t}]_{B_s \cup B_t}, \\
&\left\{ \left[\alpha_{i,b_1,b_2}\right]_{S(i,b_1,b_2)}, \quad \left[\alpha_{i,b_1,b_2} \cdot \tilde{B}_{i,b_1,b_2}\right]_{S(i,b_1,b_2)} \right\}_{i \in [n], b_1, b_2 \in \{0,1\}} .
\end{aligned}$$

Output. The obfuscator $\mathcal{O}$ outputs a circuit $\mathcal{O}(\mathsf{BP})$ that has all the handles received from the Initialization stage hardcoded into it. Given access to the oracle $\mathcal{M}$, $\mathcal{O}(\mathsf{BP})$ can add and multiply handles.

Notation. Given two handles $[\alpha]_S$ and $[\beta]_S$, we let $[\alpha]_S + [\beta]_S$ denote the handle obtained from $\mathcal{M}$ upon sending an addition query with $[\alpha]_S$ and $[\beta]_S$. Similarly, given two handles $[\alpha_1]_{S_1}$ and $[\alpha_2]_{S_2}$ such that $S_1 \cap S_2 = \emptyset$, we denote by $[\alpha_1]_{S_1} \cdot [\alpha_2]_{S_2}$ the handle obtained from $\mathcal{M}$ upon sending a multiplication query with $[\alpha_1]_{S_1}$ and $[\alpha_2]_{S_2}$. Given two matrices of handles $[M_1]_{S_1}, [M_2]_{S_2}$, we define their matrix multiplication in the natural way, and denote it by $[M_1]_{S_1} \cdot [M_2]_{S_2}$.

For input $x \in \{0,1\}^\ell$ to $\mathcal{O}(\mathsf{BP})$, and for every $i \in [n]$ let $(b_1^i, b_2^i) = (x_{\mathsf{inp}_1(i)}, x_{\mathsf{inp}_2(i)})$. On input x, $\mathcal{O}(\mathsf{BP})$ obtains the following handles:

$$h = [\tilde{\mathbf{s}}]_{B_s} \cdot \prod_{i=1}^{n} \left[\alpha_{i,b_1^i,b_2^i} \cdot \tilde{B}_{i,b_1^i,b_2^i}\right]_{S(i,b_1^i,b_2^i)} \cdot [\tilde{\mathbf{t}}]_{B_t},$$

$$h' = [\mathbf{s} \cdot \mathbf{t}]_{B_s \cup B_t} \cdot \prod_{i=1}^{n} \left[\alpha_{i,b_1^i,b_2^i}\right]_{S(i,b_1^i,b_2^i)}$$

$\mathcal{O}(\mathsf{BP})$ uses the oracle $\mathcal{M}$ to subtract the handle h' from h and performs a zero test on the result. If the zero test outputs 1 then $\mathcal{O}(\mathsf{BP})$ outputs 1, and otherwise $\mathcal{O}(\mathsf{BP})$ outputs 0.

Correctness. By construction we have that as long as none of the calls to the oracle $\mathcal{M}$ fail, subtracting the handle h' from h results in a handle to 0 if and only if:

$$\begin{aligned}
0 &= \tilde{\mathbf{s}} \cdot \prod_{i=1}^{n} \alpha_{i,b_1^i,b_2^i} \cdot \tilde{B}_{i,b_1^i,b_2^i} \cdot \tilde{\mathbf{t}} - \mathbf{s} \cdot \mathbf{t} \cdot \prod_{i=1}^{n} \alpha_{i,b_1^i,b_2^i} \\
&= \left(\tilde{\mathbf{s}} \cdot \prod_{i=1}^{n} \tilde{B}_{i,b_1^i,b_2^i} \cdot \tilde{\mathbf{t}} - \mathbf{s} \cdot \mathbf{t}\right) \cdot \prod_{i=1}^{n} \alpha_{i,b_1^i,b_2^i} \\
&= \left(\mathbf{s}^t \cdot R_0^{-1} \cdot \prod_{i=1}^{n} \left(R_{i-1} \cdot B_{i,b_1,b_2} \cdot R_i^{-1}\right) \cdot R_n^{-1} \cdot \mathbf{t} - \mathbf{s} \cdot \mathbf{t}\right) \cdot \prod_{i=1}^{n} \alpha_{i,b_1^i,b_2^i} \\
&= \mathbf{s}^t \cdot \left(\prod_{i=1}^{n} B_{i,b_1,b_2} - I_{w \times w}\right) \cdot \mathbf{t} \cdot \prod_{i=1}^{n} \alpha_{i,b_1^i,b_2^i}
\end{aligned}$$

From the definition of the branching program we have:

$$\mathsf{BP}(x) = 1 \Leftrightarrow \prod_{i=1}^{n} B_{i,b_1^i,b_2^i} = I_{w \times w}$$

Thus, if $\mathsf{BP}(x) = 1$ then $\mathcal{O}(\mathsf{BP})$ outputs 1 with probability 1. If $\mathsf{BP}(x) = 0$ then $\mathcal{O}(\mathsf{BP})$ outputs 1 with probability at most $1/p = \mathrm{negl}(n)$ over the choice of $\mathbf{s}$ and $\mathbf{t}$.

It is left to show that none of the calls to the oracle $\mathcal{M}$ fail. Note that when multiplying two matrices of handles $[M_1]_{S_1} \cdot [M_2]_{S_2}$, none of the addition or multiplication calls fail as long as $S_1 \cap S_2 = \emptyset$. Therefore, to show that none of the addition or multiplication calls to $\mathcal{M}$ fail, it is enough to show that following sets are disjoint:

$$B_s, B_t, S(1, b_1^1, b_2^1), \ldots, S(n, b_1^n, b_2^n) \ .$$

Their disjointness follows from the fact that $U_1, \ldots, U_\ell, B_s, B_t$ are disjoint, together with definition of $S(i, b_1^i, b_2^i)$ and with the fact that for every set system $\mathbb{S}^j$, for every distinct $i, i' \in \mathsf{ind}(j)$, and for every $b \in \{0,1\}$, we have that $S^j_{i,b} \cap S^j_{i',b} = \emptyset$.

To show that the zero testing call to the oracle $\mathcal{M}$ does not fail we need to show that the index set of the elements corresponding to h and h' is the entire universe. Namely, we need to show that

$$\left(\bigcup_{i=1}^{n} S(i, b_1^i, b_2^i) \right) \cup B_s \cup B_t = U \cup B_s \cup B_t \ ,$$

which follows from the following equalities:

$$\bigcup_{i=1}^{n} S(i, b_1^i, b_2^i) = \bigcup_{i=1}^{n} S^{\mathsf{inp}_1(i)}_{i,b_1^i} \cup S^{\mathsf{inp}_2(i)}_{i,b_2^i} = \bigcup_{j=1}^{\ell} \bigcup_{k \in \mathsf{ind}(j)} S^j_{k,x_i} = \bigcup_{j=1}^{\ell} U_j = U \ .$$

6 Proof of VBB in the the Ideal Graded Encoding Model

In this section we prove that the obfuscator $\mathcal{O}$ described in Section 5 is a good VBB obfuscator for $\mathbf{NC}^1$ in the ideal graded encoding model.

Let $\mathcal{C} = \{\mathcal{C}_\ell\}_{\ell \in \mathbb{N}}$ be a circuit class such that every circuit in $\mathcal{C}_\ell$ is of size $\mathsf{poly}(\ell)$ and of depth $O(\log \ell)$. We assume WLOG that all circuits in $\mathcal{C}_\ell$ are of the same depth (otherwise the circuit can be padded). It follows from Theorem 2.1 that there exist polynomial functions n and w such that on input circuit $C \in \mathcal{C}_\ell$, the branching program BP computed by $\mathcal{O}$ is of size $n(|C|)$, width $w(|C|)$, and computes on $\ell(|C|)$-bit inputs.

In Section 5 we showed that $\mathcal{O}$ satisfies the functionality requirement where the probability of $\mathcal{O}$ computing the wrong output is negligible in n. Since n is a polynomial function of $|C|$ we get that the functionality error is negligible in $|C|$, as required. It is straightforward to verify that $\mathcal{O}$ also satisfies the polynomial slowdown property. In the rest of this section we prove that $\mathcal{O}$ satisfies the virtual black-box property.

The simulator. To prove that $\mathcal{O}$ satisfies the virtual black-box property, we construct a simulator Sim that is given $1^{|C|}$, the description of an adversary $\mathcal{A}$, and oracle access to the circuit C. Sim starts by emulating the obfuscation algorithm $\mathcal{O}$. Recall that $\mathcal{O}$ converts the circuit C into a branching program BP.

However, since Sim is not given C it cannot compute the matrices B_{i,b_1,b_2} in the description of BP (note that Sim can compute the input mapping functions $\mathsf{inp}_1, \mathsf{inp}_2$ since the branching program is oblivious). Without knowing the B matrices, Sim cannot simulate the list of initial elements to the oracle $\mathcal{M}$. Instead Sim initializes $\mathcal{M}$ with formal variables.

Concretely, we extend the definition of an element to allow for values that are formal variables, as opposed to ring elements. When performing an operation $\circ$ on elements e_1, e_2 that contain formal variables, the value of the resulting element $e_1 \circ e_2$ is just the formal arithmetic expression $\alpha(e_1) \circ \alpha(e_2)$ (assuming the indexes of the elements are such that the operation is defined). We represent formal expressions as arithmetic circuits, thereby guaranteeing that the representation size remains polynomial. We say that an element is *basic* if its value is an expression that contains no gates (i.e., its just a formal variable). We say that an element e' is a *sub-element* of an element e if e was generated from e' through a sequence of operations.

To emulate $\mathcal{O}$, Sim must also emulate the oracle $\mathcal{M}$ that $\mathcal{O}$ accesses. Sim can efficiently emulate all the interfaces of $\mathcal{M}$ except for the zero testing. The problem with simulating zero tests is that Sim cannot test if the value of a formal expression is 0. Note however that the emulation of $\mathcal{O}$ does not make any zero-test queries to $\mathcal{M}$ (zero-test queries are made only by the evaluator).

When Sim completes the emulation of $\mathcal{O}$ it obtains a simulated obfuscation $\tilde{\mathcal{O}}(C)$. Sim proceeds to emulate the execution of the adversary $\mathcal{A}$ on input $\tilde{\mathcal{O}}(C)$. When $\mathcal{A}$ makes an oracle call that is not a zero test, Sim emulates $\mathcal{M}$'s answer (note that emulation of the oracle $\mathcal{M}$ is stateful and will therefore use the same handle table to emulate both $\mathcal{O}$ and $\mathcal{A}$). Since the distribution of handles generated during the simulation and during the real execution are identical, and since the simulated obfuscation $\tilde{\mathcal{O}}(C)$ consists only of handles (as opposed to elements), we have that the simulation of the obfuscation $\tilde{\mathcal{O}}(C)$ and the simulation of $\mathcal{M}$'s answers to all the queries, except for zero-test queries, is perfect.

Simulating zero testing queries. In the rest of the proof we describe how the simulator correctly simulates zero-test queries made by $\mathcal{A}$. Simulating the zero-test queries is non-trivial since the handle being tested may correspond to a formal expression whose value is unknown to Sim. (The "real" value of the formal variables depend on the circuit C). Instead we show how Sim can efficiently simulate the zero-test queries given oracle access to the circuit C.

The high-level strategy for simulating zero-test queries is as follows. Given a handle to some element, Sim tests if the value of the element is zero in two parts. In the first part, Sim decomposes the element into a sum of polynomial number of "simpler" elements that we call *single-input elements*. Each single-input element has a value that depends on a subset of the formal variables that correspond to a *specific* input to the branching program. Namely, for every single-input element there exists $x \in \{0,1\}^\ell$ such that the value of the element only depends on the formal variables in the matrices $\tilde{B}_{i,b_1^i,b_2^i}$, where $b_1^i = x_{\mathsf{inp}_1(i)}$ and $b_2^i = x_{\mathsf{inp}_2(i)}$. The main difficulty in the first step is to prove that the number of single-input elements in the decomposition is polynomial.

In the second part, Sim simulates the value of every single-input element separately. The main idea in this step is to show that the value of a single-input element for input x can be simulated only given $C(x)$. To this end, we use Kilian's proof on randomized encoding of branching programs. Unfortunately, we cannot simulate all the single-input elements at once (given oracle access to C), since their values may not be independent; in particular, they all depend on the obfuscator's randomness. Instead, we show that it is enough to zero test every single-input element individually. More concretely, we show that from every single input element that the adversary can construct, it is possible to factor out a product of the α_{i,b_1^i,b_2^i} variables. We also show that every single-input element depends on a different set of the α_{i,b_1^i,b_2^i} variables. Since the values of the α variables are chosen at random by the obfuscation, it is unlikely that the adversary makes a query where the value of two single-input elements "cancel each other" and result in a zero. Therefore, with high probability an element is zero iff it decomposes into single-input element's that are all zero individually.

Decomposition to single-input elements. Next we show that every element can be decomposed into polynomial number of single-input elements. We start by introducing some notation.

For every element e we assign an *input-profile* $\mathsf{prof}(e) \in \{0,1,*\}^\ell \cup \{\perp\}$. Intuitively, if we think of e as an intermediate element in the evaluation of the branching program on some input x, the input-profile $\mathsf{prof}(e)$ represents the partial information that can be inferred about x based on the formal variables that appear in the value of e. Formally, for every element e and for every $j \in [\ell]$, we say that the j'th bit of e's input-profile is *consistent* with the value $b \in \{0,1\}$ if e has a basic sub-element e' such that $S(e') = S(i, b_1, b_2)$ and either $j = \mathsf{inp}_1(i)$ and $b_1 = b$, or $j = \mathsf{inp}_2(i)$ and $b_2 = b$.

For every $j \in [\ell]$ and for $b \in \{0,1\}$ we set $\mathsf{prof}(e)_j = b$ if the j'th bit of e's input-profile is consistent with b but not with $1-b$. If the j'th bit of e's input-profile is not consistent with either 0 or 1 then $\mathsf{prof}(e)_j = *$. If there exist $j \in [\ell]$ such that the j'th bit of e's input-profile is consistent with both 0 and 1, then $\mathsf{prof}(e) = \perp$. In this case we say that e is *not* a single-input element and that it's profile is invalid. If $\mathsf{prof}(e) \neq \perp$ then we say that e is a single-input element. We say that an input-profile is complete if it is in $\{0,1\}^\ell$.

Next we describe an algorithm D used by Sim to decompose elements into single-input elements. Given an input element e, D outputs a set of single-input elements with distinct input-profiles such that $e = \sum_{s \in D(e)} s$, where the equality between the elements means that their values compute the same function (it does not mean that the arithmetic circuits that represent these values are identical). Note that the above requirement implies that for every $s \in D(e)$, $S(s) = S(e)$.

The decomposition algorithm D is defined recursively, as follows:

- If the input element e is basic, D outputs the singleton set $\{e\}$.
- If the input element e is of the form $e_1 + e_2$, D executes recursively and obtains the set $L = D(e_1) \cup D(e_2)$. If there exist elements $s_1, s_2 \in L$ with the same input-profile, D replaces the two elements with a single element $s_1 + s_2$. D repeats this process until all the input-profiles in L are distinct and outputs L.

– If the input element e is of the form $e_1 \cdot e_2$, D executes recursively and obtains the sets $L_1 = D(e_1), L_2 = D(e_2)$. For every $s_1 \in L_1$ and $s_2 \in L_2$, D adds the expression $s_1 \cdot s_2$ to the output set L. D then eliminates repeating input-profiles from L as described above, and outputs L.

The fact that in the above decomposition algorithm indeed $e = \sum_{s \in D(e)} s$, and that the input profiles are distinct follows from a straightforward induction. The usefulness of the above decomposition algorithm is captured by the following two claims:

Claim 6.1. *If $U \subseteq S(e)$ then all the elements in $D(e)$ are single-input elements. Namely, for every $s \in D(e)$ we have that $\mathsf{prof}(s) \neq \perp$.*

Claim 6.2. *D runs in polynomial time, and in particular, the number of elements in the output decomposition is polynomial.*

The proofs of Claims 6.1,6.2 and the formal description of how to simulate zero tests appear in the full version of this work [1].

References

[1] Barak, B., Garg, S., Kalai, Y.T., Paneth, O., Sahai, A.: Protecting obfuscation against algebraic attacks. Cryptology ePrint Archive, Report 2013/631 (2013), http://eprint.iacr.org/
[2] Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S.P., Yang, K.: On the (im)possibility of obfuscating programs. IACR Cryptology ePrint Archive 2001, 69 (2001)
[3] Barrington, D.A.: Bounded-width polynomial-size branching programs recognize exactly those languages in nc_1. In: STOC (1986)
[4] Brakerski, Z., Rothblum, G.N.: Virtual black-box obfuscation for all circuits via generic graded encoding. Cryptology ePrint Archive, Report 2013/563 (2013), http://eprint.iacr.org/
[5] Canetti, R., Vaikuntanathan, V.: Obfuscating branching programs using black-box pseudo-free groups. Cryptology ePrint Archive (2013)
[6] Coron, J.-S., Lepoint, T., Tibouchi, M.: Practical multilinear maps over the integers. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 476–493. Springer, Heidelberg (2013)
[7] Diffie, W., Hellman, M.E.: Multiuser cryptographic techniques. In: AFIPS National Computer Conference, pp. 109–112 (1976)
[8] Garg, S., Gentry, C., Halevi, S.: Candidate multilinear maps from ideal lattices. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 1–17. Springer, Heidelberg (2013)
[9] Garg, S., Gentry, C., Halevi, S., Raykova, M., Sahai, A., Waters, B.: Candidate indistinguishability obfuscation and functional encryption for all circuits. Cryptology ePrint Archive, Report 2013/451 (2013), http://eprint.iacr.org/
[10] Goyal, V., Ishai, Y., Sahai, A., Venkatesan, R., Wadia, A.: Founding cryptography on tamper-proof hardware tokens. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978, pp. 308–326. Springer, Heidelberg (2010)
[11] Hada, S.: Zero-knowledge and code obfuscation. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 443–457. Springer, Heidelberg (2000)
[12] Kilian, J.: Founding cryptography on oblivious transfer. In: Simon, J. (ed.) STOC, pp. 20–31. ACM (1988)