# Obfuscation from Semantically-Secure Multi-linear Encodings

Rafael Pass* Karn Seth† Sidharth Telang‡

November 25, 2013

**Abstract**

We define a notion of semantic security of multi-linear (a.k.a. graded) encoding schemes: roughly speaking, we require that if an algebraic attacker (obeying the multi-linear restrictions) cannot tell apart two constant-length sequences $\vec{m}_0, \vec{m}_1$ in the presence of some other elements $\vec{z}$, then encodings of these sequences should be indistinguishable. Assuming the existence of semantically secure multi-linear encodings and the LWE assumption, we demonstrate the existence of indistinguishability obfuscators for all polynomial-size circuits.

We rely on the beautiful candidate obfuscation constructions of Garg et al (FOCS'13), Brakerski and Rothblum (TCC'14) and Barak et al (ePrint'13) that were proven secure only in idealized generic multilinear encoding models, and develop new techniques for demonstrating security in the standard model, based only on semantic security of multi-linear encoding (which trivially holds in the generic multilinear encoding model).

*Cornell University, Cornell NYC Tech. Email: rafael@cs.cornell.edu. Work supported in part by a Alfred P. Sloan Fellowship, Microsoft New Faculty Fellowship, NSF Award CNS-1217821, NSF CAREER Award CCF-0746990, NSF Award CCF-1214844, AFOSR YIP Award FA9550-10-1-0093, and DARPA and AFRL under contract FA8750-11-2-0211. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US Government.

†Cornell University. Email: karn@cs.cornell.edu.

‡Cornell University. Email: sidtelang@cs.cornell.edu.

# 1 Introduction

The goal of *program obfuscation* is to "scramble" a computer program, hiding its implementation details (making it hard to "reverse-engineer"), while preserving the functionality (i.e, input/output behavior) of the program. Precisely defining what it means to "scramble" a program is non-trivial: on the one hand, we want a definition that can plausibly satisfied, on the other hand, we want a definition that is useful for applications.

A first formal definition of such program obfuscation was provided by Hada [Had00]: roughly speaking, Hada's definition—let us refer to it as *strongly virtual black-box*—is formalized using the simulation paradigm. It requires that anything an attacker can learn from the obfuscated code, could be simulated using just black-box access to the functionality.[1] Unfortunately, as noted by Hada, only learnable functionalities can satisfy such a strong notion of obfuscation: if the attacker simply outputs the code it is given, the simulator must be able to recover the code by simply querying the functionality and thus the functionality must be learnable.

An in-depth study of program obfuscation was initiated in the seminal work of Barak, Goldreich, Impagliazzo, Rudich, Sahai, Vadhan, and Yang [BGI+01]. Their central result shows that even if we consider a more relaxed simulation-based definition of program obfuscation—called *virtual black-box (VBB)* obfuscation—where the attacker is restricted to simply outputting a single bit, impossibility can still be established (assuming the existence of one-way functions). Their result is even stronger, demonstrating the existence of families of functions such that given black-box access to $f_s$ (for a randomly chosen $s$), not even a *single* bit of $s$ can be guessed with probability significantly better than 1/2, but given the code of any program that computes $f_s$, the entire secret $s$ can be recovered. Thus, even quite weak simulation-based notions of obfuscation are impossible.

But weaker notions of obfuscation may be achievable, and may still suffice for (some) applications. Indeed, Barak *et al.* [BGI+01] also suggested two such notions:

- The notion of *indistinguishability obfuscation*, first defined by Barak *et al.* [BGI+01] and explored by Garg, Gentry, Halevi, Raykova, Sahai, and Waters [GGH+13b], roughly speaking requires that obfuscations $\mathcal{O}(C_1)$ and $\mathcal{O}(C_2)$ of any two *equivalent* circuits $C_1$ and $C_2$ (i.e., whose outputs agree on all inputs) from some class $\mathcal{C}$ are computationally indistinguishable.

- The notion of *extractability obfuscation*, first defined by Barak *et al.* [BGI+01] (under the name, differing-input obfuscation) and explored by Boyle, Chung and Pass [BCP13] and by Ananth, Boneh, Garg, Sahai and Zhandry [ABG+13] strengthens the notion of indistinguishability obfuscation to also require that even if $C_1$ and $C_2$ are not equivalent circuits, if an attacker can distinguish obfuscations $\mathcal{O}(C_1)$ and $\mathcal{O}(C_2)$, then the attacker must "know" an input $x$ such that $C_1(x) \neq C_2(x)$, and this input can be efficiently "extracted" from $A$.

In a very recent breakthrough result, Garg, Gentry, Halevi, Raykova, Sahai, and Waters [GGH+13b] provided the first candidate constructions of indistinguishability obfuscators for all polynomial-size circuits so-called, based on so-called *multi-linear (a.k.a. graded) encodings*—for which candidate constructions were recently discovered in the ground-breaking work of Garg, Gentry and Halevi [GGH13a], and more recently, alternative constructions were provided by Coron, Lepoint and Tibouchi [LCT+13].

The obfuscator construction of Garg et al proceeds in two steps. They first provide a candidate construction of an indistinguishability obfuscator for $\mathsf{NC}^1$ (this construction is essentially assumed to be secure); next, they demonstrate a "bootstrapping" theorem showing how to use fully homomorphic encryption (FHE) schemes [Gen09] and indistinguishability obfuscators for $\mathsf{NC}^1$ to obtain indistinguishability obfuscators for all polynomial-size circuits.

[1] Hada actually considered a slight distributional weakening of this definition.

Further constructions of obfuscators for $\mathsf{NC}^1$ were subsequently provided by Brakerski and Rothblum [BR13] and Barak, Garg, Kalai, Paneth and Sahai [BGK$^+$13]—in fact, these definitions achieve even stronger notions of virtual-black-box obfuscation in idealized "generic" multi-linear encoding models. Additionally, Boyle, Chung and Pass [BCP13] present an alternative bootstrapping theorem, showing how to employ on extractability obfuscation for $\mathsf{NC}^1$ to obtain extractability (and thus also indistinguishability) obfuscation for both circuits and Turing machines. (Ananth et al [ABG$^+$13] also provide Turing machine extractability obfuscators but instead starting from extractability obfuscators for polynomial-size circuits.)

In parallel with the development of candidate obfuscation constructions, several surprising applications of both indistinguishability and extractability obfuscations have emerged: for instance, in the works of Garg et al [GGH$^+$13b], Sahai and Waters [SW13], Hohenberger, Sahai and Waters [HSW13], Boyle, Chung and Pass [BCP13], Boneh and Zhandry [BZ13], Garg, Gentry, Halevi and Raykova [GGHR13], Bitansky, Canetti, Paneth and Rosen [BCPR13], Boyle and Pass [BP13]. Most notable among these is the beautiful work of Sahai and Waters [SW13] (and the "punctured program" paradigm it introduces) which shows that for many of the "dream applications" of virtual black-box obfuscation (such as turning private-key encryption into public-key encryption), the weaker notion of indistinguishability obfuscation suffices.

## 1.1 Towards "Provably-Secure" Obfuscation

But despite these amazing developments, the following question remains wide open:

> *Can any "reasonable" notion of general-purpose obfuscation be achieved based on some "natural" intractability assumption?*

Note that while the construction of indistinguishability obfuscation of Garg et al is based on *some* intractability assumption, the assumption is very tightly tied to their scheme—in essence, the assumption stipulates that the scheme is a secure indistinguishability obfuscator. Rather, we are here concerned with the question of whether some *general* assumption (that is interesting in its own right, and isn't "tailored" to the scheme) can be used to obtain indistinguishability obfuscation.

The VBB constructions of Brakerski and Rothblum [BR13] and Barak et al [BGK$^+$13] give us more confidence in the plausible security of their obfuscators, in that they show that at least "generic" attacks—that treat multi-linear encoding as if they were "physical envelopes" on which multi-linear operations can be performed—cannot be used to break security of the obfuscators. But at the same time, non-generic attacks against their scheme are known—since general-purpose VBB obfuscation is impossible! Thus, it is not clear to what extent security arguments in the generic multi-linear encoding model should make us more confident that these constructions satisfy e.g., a notion of indistinguishability obfuscation.

In this work, we address the above question. We stipulate a new, but in our eyes natural, assumption regarding multi-linear encoding—the existence of, so-called, *semantically-secure multi-linear encodings*, and show how to construct indistinguishability obfuscators for $\mathsf{NC}^1$ (which then can be bootstrapped up to general circuits) based on this assumption.

## 1.2 Obfuscation From Semantically-secure Multi-linear Encodings

Recall that a multi-linear (a.k.a. graded) encoding scheme [GGH13a, GGH$^+$13b] enables anyone that has access to a *public parameter* $\mathsf{pp}$ and *encodings* $E_S^x = \mathsf{Enc}(x, S)$, $E_S^y = \mathsf{Enc}(y, S')$ of ring elements $x, y$ under the sets $S, S' \subset [n]$ to *efficiently*:[2]

[2] Just as [BR13, BGK$^+$13], we here rely on "set-based" graded encoding; these were originally called "generalized" graded encodings in [GGH13a]. Following [GGH$^+$13b, BGK$^+$13] (and in particular the notion of a "multi-linear jigsaw

- compute an encoding $E_{S \cup S'}^{x \cdot y}$ of $x \cdot y$ under the set $S \cup S'$, as long as $S \cap S' \neq \emptyset$;
- compute an encoding $E_S^{x+y}$ of $x + y$ under the set $S$ as long as $S = S'$;
- compute an encoding $E_S^{x-y}$ of $x - y$ under the set $S$ as long as $S = S'$.

(Given just access to the public-parameter $\mathsf{pp}$, generating an encoding to a particular element $x$ may not be efficient; however, it can be efficiently done given access to the *secret parameter* $\mathsf{sp}$.) Additionally, given an encoding $E_S^x$ where the set $S$ is the whole universe $[n]$, we can efficiently check whether $x = 0$ (i.e., we can "zero-test" encodings under the set $[n]$.) In essence, multi-linear encodings enable computations of certain restricted set (determined by the sets $S$ under which the elements are encoded) of arithmetic circuits, and finally determine whether the output of the circuit is 0.

Let us turn to defining a notion of semantic security for multi-linear encodings. Intuitively, given a sequence of sets $\vec{T}$ and a sequence of elements $\vec{z}$, we would want that for any two elements $m_0, m_1$ and set $S$, encodings $\mathsf{Enc}(m_0, S)$ and $\mathsf{Enc}(m_1, S)$ cannot be distinguished by any efficient attacker, that gets to see encodings $\mathsf{Enc}(\vec{z}, \vec{T})$ of $\vec{z}$ under the sets $\vec{T}$. But we can never hope that encodings of $m_0$ and $m_1$ are indistinguishable if they can already be told apart by a "algebraic" attacker that simply operates on the encodings by using the above allowed operations (in a way that respects the "set-restrictions"). Rather, our notion of *single-message* semantic security for multi-linear encodings requires that for every $S, \vec{T}$ pair, every "valid" distribution $D$ over $m_0, m_1, \vec{z}$—where a distribution $D$ is valid if no (even computationally unbounded) algebraic attacker (obeying the set-restrictions) can distinguish whether it gets access to $\{\mathsf{Enc}(m_0, S), \mathsf{Enc}(\vec{z}, \vec{T})\}$ or $\{\mathsf{Enc}(m_1, S), \mathsf{Enc}(\vec{z}, \vec{T})\}$—we have that encodings $\{\mathsf{Enc}(m_0, S), \mathsf{Enc}(\vec{z}, \vec{T})\}$ and $\{\mathsf{Enc}(m_1, S), \mathsf{Enc}(\vec{z}, \vec{T})\}$ are indistinguishable.

We will require a slight strengthening of the above notion to a *constant*-message settings, where $m_0$,$m_1$, and $S$ are replaced by *constant-length* vectors $\vec{m_0}, \vec{m_1}, \vec{S}$. In the remainder of the paper, we simply refer to constant-message semantically-secure encodings as *semantically-secure multi-linear encodings*. (For our purposes, it will in fact suffice to consider an *entropic* notion of semantic security, where we only require security to hold as long as $D$ samples $\vec{m}_0, \vec{m}_1$ and $\vec{z}$ with some high-entropy; for simplicity of exposition, we define and prove out results in the worst-case setting, but we explain in Remark 1 why our analysis also goes through if we assume the existence of just entropically secure multilinear encodings.)

Note that (randomized) multi-linear encoding scheme in the generic multi-linear encoding model of [BGK+13] are trivially semantically secure. In essence, the assumption of constant-message semantic security, stipulates that for the *particular task* of distinguishing encodings of a constant number of elements generic attacks cannot be (significantly) beaten. Our central result shows how to construct indistinguishability obfuscators for $\mathsf{NC}^1$ based on the existence of semantically-secure multi-linear encodings.

**Theorem 1** (Informally stated)**.** *Assume the existence of semantically secure multi-linear encodings. Then there exists indistinguishability obfuscators for* $\mathsf{NC}^1$.

If additionally assuming the existence of a leveled FHE [RAD78, Gen09] with decryption in $\mathsf{NC}^1$—implied e.g., by the LWE assumptions [BV11, BGV12]—this construction can be bootstrapped up to obtain indistinguishability obfuscators for all polynomial-size circuits.

**On falsifiability of semantically secure multilinear encodings** Let us point out that the assumption that a multilinear encoding scheme is semantically security is not necessarily "efficiently falsifiable" in the terminology of Naor [Nao03], since checking whether there exists some algebraic way of telling apart two constant-length sequences of element (in the presence of some other elements $z$) is not necessarily polynomial-time computable. Note, however, that the assumption that a particular scheme is

puzzles" in [GGH+13b]), we additionally enable anyone with the secret parameter to encode *any* elements (as opposed to just *random* elements as in [GGH13a]).

an indistinguishability obfuscator is not an efficiently falsifiable assumption either: a presumed attacker must exhibit two functionally-equivalent circuits $C_1$ and $C_2$ that it can distinguish obfuscations of; but checking whether two circuits are functionally equivalent may not be polynomial-time computable: in fact, assuming the existence of indistinguishability obfuscation and one-way functions it is easy to come up a method to sample $C_1$ and $C_2$ that with high probability compute different functions, yet are indistinguishable.[3]

### 1.3 Construction Overview

Following the original work of Garg et al (as well as subsequent works), our construction proceeds in three steps:

- We view the $\mathsf{NC}^1$ circuit to be obfuscated as a *branching program* $BP$ (using Barrington's Theorem [Bar86])—that is, the program is described by $m$ pairs of matrices, each one labelled with an input bit, and the program is evaluated computing by choosing one of the two matrices, based on the input, computing the product, and finally based on the product determining the output—there is a unique "accept" (i.e., output 1) matrix, and a unique "reject" (i.e., output 0) matrix.

- The branching program $BP$ is *randomized* using Kilian's technique [Kil88] (roughly, each pair of matrices is appropriately multiplied with the same random matrix $R$ while ensuring that the output is the same), and then "randomized" some more—each individual matrix is multiplied by a random *scalar* $\alpha$. Let us refer to this step as $\mathsf{Rand}$.

- Finally the randomized matrices are encoded using multilinear encodings with the sets selected appropriately. We here rely on the *straddling set* idea of [BGK+13] to determine the sets.[4] We refer to this step as $\mathsf{Encode}$.

(The original construction as well as the subsequent works also consisted of several other steps, but for our purposes these will not be needed.) The obfuscated program is now evaluated by using the multilinear operations to evaluate the branching program and finally appropriately use the zero-test to determine the output of the program. Let us refer to this construction as the "basic obfuscator".

Roughly speaking, the idea behind the basic obfuscator is that the multi-linear encodings *intuitively* ensure that any distinguisher (attacker) getting the encoding needs to multiply matrices along paths that corresponds to some input to the branching program (the straddling sets are used to ensure that the input is used consistently in the evaluation)[5]; the scalars $\alpha$ ensure that a potential distinguisher without loss of generality can use a *single* "multiplication-path" and still succeed with roughly the same probability, and finally, Kilian's randomization steps ensures that if a distinguisher *only* operates on matrices along a single path that corresponds to some input $x$ (in a consistent way), then its output can be perfectly simulated given just the output of the circuit on input $x$. (The final step relies on the fact that the output of the circuit uniquely determines product of the branching program along the path, and Kilian's randomization then ensures that the matrices along the path are random conditioned on the product being this unique value.) Thus, if a distinguisher can tell apart obfuscations of two programs $BP_0, BP_1$, there must exist some input on which they produce different outputs. The above intuitions can indeed be formalized w.r.t. *generic attackers* (that only operate on the encodings in a legal way, respecting the set restrictions), relying on arguments from [BR13, BGK+13]. However,

[3] Sample $C_1$ as the obfuscation of a "puncturable" PRF $f_s$ (as in [SW13]) with the seed $s$ hard-coded, and $C_2$ as the obfuscation of a "punctured" version of $f_s$ where a random input $x$ has been changed to a random output $y$. With high probability, $C_1$ and $C_2$ differ on $x$, yet following the hybrid argument in [SW13] it is easy to see that the circuits cannot be distinguished.

[4] Although we have not verified all the details, to achieve indistinguishability obfuscation, it seems that we could have also relied on the original encoding methods, but using straddling sets simplifies the analysis.

[5] The encodings, however, still permit an attacker to add elements within matrices.

although security w.r.t. generic attackers will be useful to us (as we shall see shortly), we are interested in proving security w.r.t. *all polynomial-size attackers*.

Towards this, we will add an additional program transformation steps before the Rand and Encode steps. Roughly speaking, we would like to have a method $\mathsf{Merge}(BP_0, BP_1, b)$ that "merges" $BP_0$ and $BP_1$ into a single branching program that evaluates $BP_b$; additionally, we require that $\mathsf{Merge}(BP_0, BP_1, 0)$ and $\mathsf{Merge}(BP_0, BP_1, 1)$ only differ in a constant (four will suffice) number of matrices. We achieve this merge procedure by connecting together $BP_0, BP_1$ into a branching program of double width and adding two "switch" matrices in the beginning and the end, determining if we should go "up" or "down". Thus, to switch between $\mathsf{Merge}(BP_0, BP_1, 0)$ (which is functionally equivalent to $BP_0$) and $\mathsf{Merge}(BP_0, BP_1, 1)$ (which is functionally equivalent to $BP_1$) we just need to switch the "switch matrices". Our candidate obfuscator is now defined as $i\mathcal{O}(B) = \mathsf{Encode}(\mathsf{Rand}(\mathsf{Merge}(BP, I, 0)))$, where $I$ is simply a "dummy" program of the same size as $BP$.[6]

The idea behind the merge procedure is that to prove that obfuscations of two programs $BP_0$, $BP_1$ are indistinguishable, we can come up with a sequence of hybrid experiments that start with $i\mathcal{O}(BP_0)$ and end with $i\mathcal{O}(BP_1)$, but between any two hybrid only change a constant number of encodings, and thus intuitively we may rely on semantic security of multilinear encodings to formalize the above intuitions. At a high level, our strategy will be to matrix-by-matrix, replace the dummy branching program in the obfuscation of $BP_0$ with the branching program for $BP_1$. Once the entire dummy branching program has been replaced by $BP_1$, we flip the "switch" so that the composite branching program now computes the branching program for $BP_1$. We then replace the branching program for $BP_0$ with $BP_1$, matrix by matrix, so that we have two copies of the branching program for $BP_1$. We now flip the "switch" again, and finally restore the dummy branching program, so that we end up with one copy of $BP_1$ and one copy of the dummy, which is now a valid obfuscation of $BP_1$. In this way, we transition from an obfuscation of $BP_0$ to an obfuscation of $BP_1$, while only changing a small piece of the obfuscation in each step.

More precisely, consider the following sequence of hybrids.

- We start off with $i\mathcal{O}(BP_0) = \mathsf{Enc}(\mathsf{Rand}(\mathsf{Merge}(BP_0, I, 0)))$
- We consider a sequence of hybrids where we gradually change the dummy program $I$ to become $BP_1$; that is, we consider $\mathsf{Encode}(\mathsf{Rand}(\mathsf{Merge}(BP_0, BP', 0)))$, where $BP'$ is "step-wise" being populated with elements from $BP_1$.
- We reach $\mathsf{Encode}(\mathsf{Rand}(\mathsf{Merge}(BP_0, BP_1, 0)))$.
- We turn the "switch" : $\mathsf{Encode}(\mathsf{Rand}(\mathsf{Merge}(BP_0, BP_1, 1)))$.
- We consider a sequence of hybrids where we gradually change the $BP_0$ to become $BP_1$; that is, we consider $\mathsf{Encode}(\mathsf{Rand}(\mathsf{Merge}(BP', BP_1, 1)))$, where $BP'$ is "step-wise" being populated with elements from $BP_1$.
- We reach $\mathsf{Encode}(\mathsf{Rand}(\mathsf{Merge}(BP_1, BP_1, 1)))$.
- We turn the "switch" back: $\mathsf{Encode}(\mathsf{Rand}(\mathsf{Merge}(BP_1, BP_1, 0)))$.
- We consider a sequence of hybrids where we gradually change the second $BP_1$ to become $I$; that is, we consider $\mathsf{Encode}(\mathsf{Rand}(\mathsf{Merge}(BP_1, BP', 0)))$, where $BP'$ is "step-wise" being populated with elements from $I$.
- We reach $\mathsf{Encode}(\mathsf{Rand}(\mathsf{Merge}(BP_1, I, 0))) = i\mathcal{O}(BP_1)$.

[6]This description oversimplifies a bit. Formally, the Rand step needs to depends on the field size used in the Encode steps, and thus in our formal treatment we combine these two steps together.

By construction we have that if $BP_0$ and $BP_1$ are functionally equivalent, then so will all the hybrid programs–the key point is that we only "morph" between two branching programs on the "inactive" part of the merged branching program. Furthermore, by construction, between any two hybrids we only change a constant number of elements. Thus, if some distinguisher can tell apart $i\mathcal{O}(BP_0)$ and $i\mathcal{O}(BP_1)$, it must be able to tell apart two consecutive hybrids. But, by semantic security it then follows that some *algebraic* attacker can tell apart the encodings in the two hybrids. Roughly speaking, we can now rely on indistinguishable security of the basic obfuscator w.r.t. to just *generic* attackers to complete the argument.

There is a catch with the final step though. Recall that to rely on Kilian's simulation argument it was crucial that there are unique accept and reject matrices. For our "merged" programs, this is no longer the case (the output matrix is also a function of the second "dummy" program). We overcome this issue by noting that the first column of the output matrix actually is unique, and this is all we need to determine the output of the branching program. Consequently it suffices to release encodings of the *just* first column (as opposed to the whole matrices) of the last matrix pair in the branching program, and we can still determine the output of the branching program. As we show, for such a modified scheme, we can also simulate the (randomized) matrices along an "input-path" given just the first column of the output matrix. This concludes the description of our indistinguishability obfuscator.

## 2 Preliminaries

Let $\mathbb{N}$ denote the set of positive integers, and $[n]$ denote the set $\{1, 2, \ldots, n\}$. Let $\mathbb{Z}$ denote the integers, and $\mathbb{Z}_p$ the integers modulo $p$. Given a string $x$, we let $x[i]$, or equivalently $x_i$, denote the $i$-th bit of $x$. For a matrix $M$, we let $M[i, j]$ denote the entry of $M$ in the $i$th row and $j$th column. We use $\mathbf{e}_k$ to denote the vector that is 1 in position $k$, and 0 in all other positions. The length of $\mathbf{e}_k$ is generally clear from the context. We use $I_{w\times w}$ to denote the identity matrix with dimension $w \times w$.

By a probabilistic algorithm we mean a Turing machine that receives an auxiliary random tape as input. If $M$ is a probabilistic algorithm, then for any input $x$, $M(x)$ represents the distribution of outputs of $M(x)$ when the random tape is chosen uniformly. An oracle algorithm $M^O$ is a machine $M$ that gets oracle access to another machine $O$, that is, it can access $O$'s functionality as a black-box.

By $x \leftarrow S$, we denote an element $x$ is sampled from a distribution $S$. If $F$ is a finite set, then $x \leftarrow F$ means $x$ is sampled uniformly from the set $F$. To denote the ordered sequence in which the experiments happen we use semicolon, e.g. $(x \leftarrow S; (y, z) \leftarrow A(x))$. Using this notation we can describe probability of events. For example, if $p(\cdot, \cdot)$ denotes a predicate, then $\Pr[x \leftarrow S; (y, z) \leftarrow A(x) : p(y, z)]$ is the probability that the predicate $p(y, z)$ is true in the ordered sequence of experiments $(x \leftarrow S; (y, z) \leftarrow A(x))$. The notation $\{(x \leftarrow S; (y, z) \leftarrow A(x) : (y, z))\}$ denotes the resulting probability distribution $\{(y, z)\}$ generated by the ordered sequence of experiments $(x \leftarrow S; (y, z) \leftarrow A(x))$. We define the support of a distribution $\mathsf{supp}(S)$ to be $\{y : \Pr[x \leftarrow S : x = y] > 0\}$.

### 2.1 Obfuscation

We recall the definition of indistinguishability obfuscation due to [BGI+01].

**Definition 1** (Indistinguishability Obfuscator)**.** *A uniform PPT machine $i\mathcal{O}$ is an* indistinguishability obfuscator *for a class of circuits $\{\mathcal{C}_n\}_{n\in\mathbb{N}}$ if the following conditions are satisfied*

- **Correctness:** *There exists a negligible function $\varepsilon$ such that for every $n \in \mathbb{N}$, for all $C \in \mathcal{C}_n$, for all inputs $x \in \{0, 1\}^n$, we have*

$$\Pr[C' \leftarrow i\mathcal{O}(C_n) : C'(x) = C_n(x)] = 1 - \varepsilon(n).$$

- **Security:** *For every pair of circuit ensembles* $\{C_n^0\}_{n\in\mathbb{N}}$ *and* $\{C_n^1\}_{n\in\mathbb{N}}$ *such that for all* $n \in \mathbb{N}$*, for every pair of circuits* $C_n^0, C_n^1 \in \mathcal{C}_n$ *such that* $C_n^0(x) = C_n^1(x)$ *for all* $x \in \{0,1\}^n$ *the following holds: For every nuPPT adversary* $A$ *there exists a negligible function* $\varepsilon$ *such that for all* $n \in \mathbb{N}$*,*

$$|Pr[C' \leftarrow i\mathcal{O}(C_n^0) : A(C') = 1] - Pr[C' \leftarrow i\mathcal{O}(C_n^1) : A(C') = 1]| \leq \varepsilon(n)$$

## 2.2 Graded Encoding Schemes

We proceed to defining graded (multilinear) encoding schemes, originally introduced by Garg, Gentry and Halevi [GGH13a]. Just as [BR13, BGK+13], we here rely on "set-based" graded encoding; these were originally called "generalized" graded encodings in [GGH13a]. Following [GGH+13b, BGK+13] and the notion of "multi-linear jigsaw puzzles" from [GGH+13b], we additionally enable anyone with the secret parameter to encode *any* elements (as opposed to just *random* elements as in [GGH13a]).

**Definition 2** (($k, R$)-Graded Encoding Scheme)**.** *A* $(k, R)$*-graded encoding scheme for* $k \in \mathbb{N}$ *and ring* $R$ *is a collection of sets* $\{E_S^\alpha : \alpha \in R, S \subseteq [k]\}$ *with the following properties*

- *For every* $S \subseteq [k]$ *the sets* $\{E_S^\alpha : a \in R\}$ *are disjoint.*
- *There are associative binary operations* $\oplus$ *and* $\ominus$ *such that for every* $\alpha_1, \alpha_2 \in R$*,* $S \subseteq [k]$*,* $u_1 \in E_S^{\alpha_1}$ *and* $u_2 \in E_S^{\alpha_2}$ *it holds that* $u_1 \oplus u_2 \in E_S^{\alpha_1+\alpha_2}$ *and* $u_1 \ominus u_2 \in E_S^{\alpha_1-\alpha_2}$ *where '*$+$*' and '*$-$*' are the addition and subtraction operations in* $R$*.*
- *There is an associative binary operation* $\otimes$ *such that for every* $\alpha_1, \alpha_2 \in R$*,* $S_1, S_2 \subseteq [k]$ *such that* $S_1 \cap S_2 = \emptyset$*,* $u_1 \in E_{S_1}^{\alpha_1}$ *and* $u_2 \in E_{S_2}^{\alpha_2}$ *it holds that* $u_1 \otimes u_2 \in E_{S_1\cup S_2}^{\alpha_1\cdot\alpha_2}$ *where '*$\cdot$*' is multiplication in* $R$*.*

**Definition 3** (Graded Encoded Scheme)**.** *A graded encoding scheme* $\mathcal{E}$ *is associated with a tuple of PPT algorithms,* $(\mathsf{InstGen}_\mathcal{E}, \mathsf{Enc}_\mathcal{E}, \mathsf{Add}_\mathcal{E}, \mathsf{Neg}_\mathcal{E}, \mathsf{Mult}_\mathcal{E}, \mathsf{isZero})_\mathcal{E}$ *which behave as follows:*

- *Instance Generation:* $\mathsf{InstGen}_\mathcal{E}$ *takes as input the security parameter* $1^n$ *and multi-linearity parameter* $1^k$*, and outputs secret parameters* $\mathsf{sp}$ *and public parameters* $\mathsf{pp}$ *which describe a* $(k, R)$*-graded encoding scheme* $\{E_S^\alpha : \alpha \in R, S \subseteq [k]\}$*. We refer to* $E_S^\alpha$ *as the set of encodings of the pair* $(\alpha, S)$*. In this work we consider graded encoding schemes where* $R$ *is* $\mathbb{Z}_p$ *and* $p$ *is a prime exponential in* $n$*.*
- *Encoding:* $\mathsf{Enc}_\mathcal{E}$ *takes as input the secret parameters* $\mathsf{sp}$*, an element* $\alpha \in R$ *and set* $S \subseteq [k]$*, and outputs a random encoding of the pair* $(\alpha, S)$*.*
- *Addition:* $\mathsf{Add}_\mathcal{E}$ *takes as input the public parameters* $\mathsf{pp}$ *and encodings* $u_1 \in E_{S_1}^{\alpha_1}, u_2 \in E_{S_2}^{\alpha_2}$*, and outputs an encoding of the pair* $(\alpha_1 + \alpha_2, S)$ *if* $S_1 = S_2 = S$ *and outputs* $\perp$ *otherwise.*
- *Negation:* $\mathsf{Neg}_\mathcal{E}$ *takes as input the public parameters* $\mathsf{pp}$ *and encodings* $u_1 \in E_{S_1}^{\alpha_1}, u_2 \in E_{S_2}^{\alpha_2}$*, and outputs an encoding of the pair* $(\alpha_1 - \alpha_2, S)$ *if* $S_1 = S_2 = S$ *and outputs* $\perp$ *otherwise.*
- *Multiplication:* $\mathsf{Mult}_\mathcal{E}$ *takes as input the the public parameters* $\mathsf{pp}$ *and encodings* $u_1 \in E_{S_1}^{\alpha_1}, u_2 \in E_{S_2}^{\alpha_2}$*, and outputs an encoding of the pair* $(\alpha_1 \cdot \alpha_2, S_1 \cup S_2)$ *if* $S_1 \cap S_2 = \emptyset$ *and outputs* $\perp$ *otherwise.*
- *Zero testing:* $\mathsf{isZero}_\mathcal{E}$ *takes as input the public parameters* $\mathsf{pp}$ *and an encoding* $u \in E_S(\alpha)$*, and outputs 1 if and only if* $\alpha = 0$ *and* $S$ *is the universe set* $[k]$*.*

*Whenever it is clear from the context, to simplify notation we drop the subscript* $\mathcal{E}$ *when we refer to the above procedures (and simply call them* $\mathsf{InstGen}, \mathsf{Enc}, \ldots$*).*

Note that the above procedures allow algebraic operations on the encodings in a restricted way. Given the public parameters and encodings made under the sets $\vec{S}$, one can only perform algebraic operations that are allowed by the structure of the sets in $\vec{S}$. We call such operations $\vec{S}$-respecting and formalize this notion as follows:

**Definition 4** (Set Respecting Arithmetic Circuits)**.** *For any ring $R$, $k \in \mathbb{N}$ and $\vec{S} \in (2^{[k]})^n$, we say that an arithmetic circuit $C$ (i.e. gates perform only ring operations $\{+, -, \cdot\}$) of input size $n$ is $\vec{S}$-*respecting *if it holds that*

- *We tag every input wire of $C$ with the corresponding set in $\vec{S}$. The $i^{th}$ input wire is tagged with $\vec{S}[i]$.*
- *For every $+$ and $-$ gate in $C$, if the tags of the two input wires are the same set $S$ then the output wire of the gate is tagged with $S$. Otherwise the output wire is tagged with $\perp$.*
- *For every $\cdot$ gate in $C$, if the tags of the two input wires are sets $S_1$ and $S_2$ and $S_1 \cap S_2 = \emptyset$ then the output wire of the gate is tagged with $S_1 \cup S_2$. Otherwise the output wire is tagged with $\perp$.*
- *It holds that the output wire is tagged with the universe set $[k]$.*

The following lemma is a simple corollary of the efficient procedures described in Definition 4. It states that given the public parameters and some encodings made under the sets $\vec{S}$, one can efficiently zero test the result of any $\vec{S}$ respecting arithmetic circuit on the elements underlying the encodings.

**Lemma 2** (Correctness)**.** *Let $\mathcal{E}$ be a graded encoding scheme. There exists a PPT* Eval *such that for any $k, n, m \in \mathbb{N}$,* Eval *takes as input the public parameters* $\mathsf{pp} \in \mathsf{InstGen}(1^n, 1^k)$ *that describe a $(k, R)$-graded encoding scheme, a sequence of encodings of some ring elements under some sets $\{u_i\}_{i=1}^m$ where $u_i \in E_{S_i}^{\alpha_i}$, $\alpha_i$ is a ring element and $S_i \subseteq [k]$, and any $\vec{S} = \{S_i\}_{i=1}^m$-respecting arithmetic circuit $C$, and outputs 1 if and only if $C(\{\alpha_i\}_{i=1}^m) = 0$.*

### 2.3 Branching programs for $\mathsf{NC}^1$

**Definition 5** (Oblivious Matrix Branching Program)**.** *A* branching program *of width $w$ and length $m$ for $n$-bit inputs is given by a sequence:*

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

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

$$BP(x) \stackrel{def}{=} \begin{cases} 1, & \text{if } (\prod_{i=1}^m B_{i,x[\mathsf{inp}(i)]}) \cdot \mathbf{e}_1 = \mathbf{e}_1. \\ 0, & \text{otherwise} \end{cases}$$

*The branching program is said to be* oblivious *if $\mathsf{inp} : [m] \to [n]$ is a fixed function, independent of the function being evaluated. The branching program is said to have fixed accept and reject matrices $\mathsf{P}_{\mathsf{accept}}$ and $\mathsf{P}_{\mathsf{reject}}$ if, for all $x \in \{0,1\}^n$,*

$$\prod_{i=1}^m B_{i,x[\mathsf{inp}(i)]} = \begin{cases} \mathsf{P}_{\mathsf{accept}} & \text{when } BP(x) = 1 \\ \mathsf{P}_{\mathsf{reject}} & \text{when } BP(x) = 0 \end{cases}$$

In other words, the branching program accepts $x$ whenever $\prod_{i=1}^m B_{i,x[\mathsf{inp}(i)]}$ is a matrix for a permutation that fixes the first element, and rejects $x$ otherwise.
In particular, we still have the following theorem due to Barrington:

**Theorem 3.** *([Bar86]) For any depth $d$ and input length $n$, there exists a length $m = 4^d$, a labeling function* $\mathsf{inp} : [m] \to [n]$*, an accepting permutation* $\mathsf{P}_{\mathsf{accept}}$ *with* $\mathsf{P}_{\mathsf{accept}} \cdot \mathbf{e}_1 = \mathbf{e}_1$*, and a rejecting permutation* $\mathsf{P}_{\mathsf{reject}}$ *with* $\mathsf{P}_{\mathsf{reject}} \cdot \mathbf{e}_1 = \mathbf{e}_k$ *where* $k \neq 1$ *such that, for every fan-in 2 boolean circuit $C$ of depth $d$ and $n$ input bits, there exists an oblivious matrix branching program* $BP = \{\mathsf{inp}(i), B_{i,0}, B_{i,1}\}_{i=1}^m$*, of width 5 and length $m$ that computes the same function as the circuit $C$.*

In particular, every circuit in $\mathsf{NC}^1$ has a polynomial length branching program of width 5. Further, two circuits of the same depth $d$ will have fixed accepting and rejecting permutations $\mathsf{P}_{\mathsf{accept}}$ and $\mathsf{P}_{\mathsf{reject}}$, and a fixed labelling function $\mathsf{inp} : [m] \to [n]$.

# 3 Semantically Secure Graded Encoding Schemes

In this section we define what it means for a graded encoding scheme to be semantically secure. Intuitively, we want to say that such an encoding scheme is like semantically secure encryption; the encodings of any two messages $m_0$ and $m_1$ should be indistinguishable. However, as previously mentioned, graded encoding schemes enable certain "set-restricted" computations on the encodings, and zero-testing. Given this ability it may be trivial to distinguish encodings of $m_0$ and $m_1$ given some auxiliary encodings of additional elements. Rather, we require that for every $S, \vec{T}$ pair, every "valid" distribution $D$ over $m_0, m_1, \vec{z}$—where a distribution $D$ is valid if no algebraic attacker, even computionally unbounded ones, obeying the set-restrictions, can distinguish whether it gets access to $\{\mathsf{Enc}(m_0, S), \mathsf{Enc}(\vec{z}, \vec{T})\}$ or $\{\mathsf{Enc}(m_1, S), \mathsf{Enc}(\vec{z}, \vec{T})\}$—we have that encodings $\{\mathsf{Enc}(m_0, S), \mathsf{Enc}(\vec{z}, \vec{T})\}$ and $\{\mathsf{Enc}(m_1, S), \mathsf{Enc}(\vec{z}, \vec{T})\}$ are indistinguishable. What this assumption guarantees is that no adversary can gain an advantage by acting non-algebraically.

As mentioned before, we focus on a *constant*-message settings, where $m_0$,$m_1$, and $S$ are replaced by *constant-length* vectors $\vec{m_0}, \vec{m_1}, \vec{S}$. On the other hand, for our purposes, it will suffice to consider an *entropic* notion of semantic security, where we only require security of the graded encoded scheme to hold as long as $D$ samples $\vec{m}_0, \vec{m}_1$ and $\vec{z}$ with some high-entropy.

We start by formally defining an *algebraic adversary.* Such an adversary, when given a set of encodings, is restricted to only the public efficient procedures of the graded encoding scheme. That is, it can only homomorphically evaluate certain algebraic operations on the encoded elements (restricted by the sets under which the encodings are made), and can check whether an element encoded under the universe set is zero or not. We formalize this restriction by considering adversaries that interact with the following oracle.

**Definition 6** (Oracle $\mathcal{M}$)**.** *Let $\mathcal{M}$ be an oracle which operates as follows:*

- *$\mathcal{M}$ gets as initial input a ring $R$, $k \in \mathbb{N}$ and list $L$ of $m$ pairs $\{(\alpha_i, S_i)\}_{i=1}^m$, $\alpha \in R$ and $S \subseteq [k]$.*
- *Every oracle query to $\mathcal{M}$ is an arithmetic circuit $C : R^m \to R$. When queried with $C$, $\mathcal{M}$ checks whether $C$ is a $\vec{S}$-respecting arithmetic circuit where $\vec{S} = \{S_i\}_{i=1}^m$. If not, $\mathcal{M}$ outputs $\perp$. Otherwise, $\mathcal{M}$ computes $C$ on $\{\alpha_i\}_{i=1}^m$ and outputs 1 if and only if the output of $C$ is zero, and outputs 0 otherwise.*

We next formalize what it means for a distribution over $(\vec{m_0}, \vec{m_1}, \vec{z})$ to be *valid* with respect to the sets $(\vec{S}, \vec{T})$. Intuitively, we say that a distribution is valid if no algebraic adversary can distinguish the encodings of $(\vec{m_0}, \vec{z})$ and $(\vec{m_1}, \vec{z})$, under the sets $(\vec{S}, \vec{T})$. We define such a distribution through the notion of a $(\vec{S}, \vec{T})$-respecting message sampler.

**Definition 7** (Respecting Message Sampler)**.** *Let $\mathcal{E}$ be a graded encoding scheme, $q(\cdot)$, $k(\cdot)$ and $\mu(\cdot)$ be polynomials, $c \in \mathbb{N}$ be a constant and $\{(\vec{S_n}, \vec{T_n})\}_{n\in\mathbb{N}}$ be an ensemble where $\vec{S_n}$ is a sequence of $c$ sets and $\vec{T_n}$ is a sequence of $q(n)$ sets $\subseteq [k]$. We say that a nuPPT $M$ is a* $(q, k, c, \{(\vec{S_n}, \vec{T_n})\}_{n\in\mathbb{N}})$-respecting message sampler *if*

- *$M$ takes as input the public parameters of a graded encoding scheme* $\mathsf{pp} \in \mathsf{InstGen}(1^n, 1^{k(n)})$ *which describes a ring $R$ and outputs*
  - *a pair of sequences of $c$ ring elements, $\vec{m_0}$ and $\vec{m_1}$ and*
  - *a sequence of $q(n)$ ring elements $\vec{z}$.*
- *For every (computationally unbounded) oracle machine $A$ that makes at most polynomially many oracle queries*[7] *(called the* algebraic adversary*) there exists a negligible function $\varepsilon$ such that for every security parameter $n \in \mathbb{N}$,*

$$|Pr[(\mathsf{sp}, \mathsf{pp}) \leftarrow \mathsf{InstGen}(1^n, 1^{k(n)}), (\vec{m_0}, \vec{m_1}, \vec{z}) \leftarrow M(\mathsf{pp}) : A^{\mathcal{M}(\mathsf{pp}, \vec{p_0})}(1^n) = 1] -$$
$$Pr[(\mathsf{sp}, \mathsf{pp}) \leftarrow \mathsf{InstGen}(1^n, 1^{k(n)}), (\vec{m_0}, \vec{m_1}, \vec{z}) \leftarrow M(\mathsf{pp}) : A^{\mathcal{M}(\mathsf{pp}, \vec{p_1})}(1^n) = 1]| \leq \varepsilon(n)$$

*where* $\vec{p_b} = \{(m_b[i], S_i)\}_{i=1}^{c}, \{(z[i], T_i)\}_{i=1}^{q(n)}$.

We now define what it means for a graded encoding scheme to be semantically secure. Roughly speaking, we require that for any sets $(\vec{S}, \vec{T})$, and any $(\vec{S}, \vec{T})$-respecting message sampler, encodings of $(\vec{m_0}, \vec{z})$ and $(\vec{m_1}, \vec{z})$ under the sets $(\vec{S}, \vec{T})$ are indistinguishable, when $(\vec{m_0}, \vec{m_1}, \vec{z})$ is sampled by the message sampler.

**Definition 8** (Semantic Security)**.** *We say a graded encoding scheme $\mathcal{E}$ is* semantically secure *if for every every polynomials $q(\cdot)$ and $k(\cdot)$, constant $c \in \mathbb{N}$, every ensemble $\{(\vec{S_n}, \vec{T_n})\}_{n\in\mathbb{N}}$ where $\vec{S_n} \subseteq [k(n)]^c$ and $\vec{T_n} \subseteq [k(n)]^{q(n)}$, every $(q, k, c, \{\vec{S_n}\}_{n\in\mathbb{N}})$-respecting message sampler $M$ and nuPPT adversary $A$, there exists a negligible function $\epsilon$ such that for every security parameter $n \in \mathbb{N}$,*

$$|Pr[\boldsymbol{Output}_0(q, k, c, A, M, n, (\vec{S_n}, \vec{T_n})) = 1] - Pr[\boldsymbol{Output}_1(q, k, c, A, M, n, (\vec{S_n}, \vec{T_n})) = 1]| \leq \epsilon(n)$$

*where $\boldsymbol{Output}_b(q, k, c, A, M, n, (\vec{S_n}, \vec{T_n}))$ is $A$'s output in the following game.*

- *Run* $(\mathsf{sp}, \mathsf{pp}) \leftarrow \mathsf{InstGen}(1^n, 1^{k(n)})$.
- *$M$ takes as input the public parameters* $\mathsf{pp}$ *and outputs $\vec{m_0}$, $\vec{m_1}$ and $\vec{z}$.*
- *Encode each element of $\vec{m_b}$ and $\vec{z}$ with the corresponding set in $\vec{S_n}$ and $\vec{T_n}$. That is, compute the following encodings*

$$\vec{u_b} \leftarrow \{\mathsf{Enc}(\mathsf{sp}, \vec{m_0}[i], \vec{S_n}[i])\}_{i=1}^{c}, \{\mathsf{Enc}(\mathsf{sp}, \vec{z}[i], \vec{T_n}[i])\}_{i=1}^{q(n)}$$

- *$A$ takes as input $\vec{u_b}$ and outputs a bit $b' \in \{0, 1\}$*

# 4 Construction of an Indistinguishability Obfuscator

In this section, we describe our construction of an indistinguishability obfuscator $i\mathcal{O}$. We will prove its security in Section 5, based on the security notions defined above.

As in previous works [GGH$^+$13b, BR13, BGK$^+$13], the strategy for our construction will be to convert an $\mathsf{NC}^1$ circuit into an oblivious matrix branching program, apply Kilian's randomization technique to the matrices, and then encode these matrices using the graded encoding scheme. The encoding will be using a so-called "straddling set system" (as in [BGK$^+$13]) that will enforce that any arithmetic circuit operating on these encodings can be decomposed into a sum of terms such that each term can be expressed using only encodings that come from one branch of the branching program (more specifically,

[7]Our proofs work even if the algebraic adversary $A$ makes only subexponentially many oracle queries.

from the path through the branching program corresponding to evaluating a particular input $x$ to the branching program).

The biggest change from previous work is that before randomizing and encoding the branching program, we double its width by chaining a dummy branching program to it that computes the constant 1, and then add a branch at the very start that chooses whether to use the true program or the dummy, based on a "switch".

At a high level, to show indistinguishability of obfuscations of $C_1$ and $C_2$, our strategy will be to obfuscate the branching program for $C_1$ together with the dummy, and then, matrix by matrix, replace the dummy branching program with the branching program for $C_2$. Once the entire dummy branching program has been replaced by $C_2$, we flip the "switch" so that the composite branching program now computes the branching program for $C_2$. We then replace the branching program for $C_1$ with $C_2$, matrix by matrix, so that we have two copies of the branching program for $C_2$. We now flip the "switch" again, and finally restore the dummy branching program, so that we end up with one copy of $C_2$ and one copy of the dummy.

In this way, we transition from an obfuscation of $C_1$ to an obfuscation of $C_2$, while only changing a small piece of the obfuscation in each step, namely a single level of the underlying branching program. We will later show, in the following section, that each step of the transitions must be indistinguishable based on our hardness assumption. In particular, we show that no algebraic adversary can distinguish between two hybrids, and thus the two distributions should be computationally indistinguishable based on our assumption.

## 4.1 Merging Branching Programs

We now describe a method Merge for combining two branching programs together to create a composite branching program of double width, in a way that enables switching by changing only a small number of matrices.

**Construction 1** (Merging branching programs)**.** *Let $BP_0 = \{\mathsf{inp}(i), B^0_{i,0}, B^0_{i,1}\}_{i=1}^m$ and $BP_1 = \{\mathsf{inp}(i), B^1_{i,0}, B^1_{i,1}\}_{i=1}^m$ be oblivious matrix branching programs, each of width $w$ and length $m$ for $n$ input bits. (We assume that the same labelling function $\mathsf{inp} : [m] \to [n]$ is used for each of $BP_0$ and $BP_1$.)*
*Define branching programs $\hat{BP}_0 = \{\mathsf{inp}'(i), \hat{B}^0_{i,0}, \hat{B}^0_{i,1}\}_{i=1}^{m+2}$ and $\hat{BP}_1 = \{\mathsf{inp}'(i), \hat{B}^1_{i,0}, \hat{B}^1_{i,1}\}_{i=1}^{m+2}$, each of width $2w$ and length $m+2$ on $l$ input bits, where:*

$$\mathsf{inp}'(i) \stackrel{def}{=} \begin{cases} 1, & \text{when } i = 1 \\ \mathsf{inp}(i-1), & \text{when } 2 \le i \le m+1 \\ 1, & \text{when } i = m+2 \end{cases}$$

*and, for all levels except the first and the last, $\hat{BP}_0$ and $\hat{BP}_1$ agree, given by:*

$$\hat{B}^0_{i,b} = \hat{B}^1_{i,b} \stackrel{def}{=} \begin{pmatrix} B^0_{(i-1),b} & 0 \\ 0 & B^1_{(i-1),b} \end{pmatrix} \quad \text{for all } 2 \le i \le m+1 \text{ and } b \in \{0,1\}$$

*and the first and last levels are given by:*

$$\hat{B}^0_{1,b} = \hat{B}^0_{m+2,b} = I_{2w\times 2w} \qquad \text{for } b \in \{0,1\}$$

$$\hat{B}^1_{1,b} = \hat{B}^1_{m+2,b} = \begin{pmatrix} 0 & I_{w\times w} \\ I_{w\times w} & 0 \end{pmatrix} \qquad \text{for } b \in \{0,1\}$$

*We define* Merge *so that* $\mathsf{Merge}(BP_0, BP_1, 0) = \hat{BP}_0$ *and* $\mathsf{Merge}(BP_0, BP_1, 1) = \hat{BP}_1$.

We will show that $\hat{BP}_0$ and $\hat{BP}_1$ are matrix branching programs that compute the same functions as $BP_0$ and $BP_1$ respectively, with the additional feature that $\hat{BP}_0$ and $\hat{BP}_1$ differ from each other in only two levels, namely the first and the last. Further, since $\mathsf{inp}'$ does not depend on the function being computed, $\hat{BP}_0$ and $\hat{BP}_1$ are *oblivious* matrix branching programs.

Accordingly, with respect to $\mathsf{Merge}(BP_0, BP_1, b)$ we will often use the phrase *active branching program* to refer to $BP_b$.

**Claim 4.** *For $BP_0 = \{\mathsf{inp}(i),\ B^0_{i,0}, B^0_{i,1}\}_{i=1}^m$ and $BP_1 = \{\mathsf{inp}(i), B^1_{i,0}, B^1_{i,1}\}_{i=1}^m$ each of width $w$ and length $m$ on $n$ input bits, define $\hat{BP}_0$ and $\hat{BP}_1$ as above. Then, for each $b \in \{0,1\}$, $x \in \{0,1\}^n$,*

$$\prod_{i=1}^{m+2} \widehat{B}^b_{i,x[\mathsf{inp}'(i)]} = \begin{pmatrix} \prod_{i=1}^m B^b_{i,x[\mathsf{inp}(i)]} & 0 \\ 0 & \prod_{i=1}^m B^{1-b}_{i,x[\mathsf{inp}(i)]} \end{pmatrix}$$

*Proof.* We observe that $\hat{BP}_0$ and $\hat{BP}_1$ agree on each level except the first and last, that is,

$$\widehat{B}^0_{i,b} = \widehat{B}^1_{i,b} = \begin{pmatrix} B^0_{(i-1),b} & 0 \\ 0 & B^1_{(i-1),b} \end{pmatrix} \qquad \forall \quad i : 2 \le i \le m+1, \quad b \in \{0,1\}$$

Then we have, for any $x \in \{0,1\}^n$,

$$\begin{aligned}\prod_{i=2}^{m+1} \widehat{B}^0_{i,x[\mathsf{inp}'(i)]} = \prod_{i=2}^{m+1} \widehat{B}^1_{i,x[\mathsf{inp}'(i)]} &= \prod_{i=2}^{m+1} \begin{pmatrix} B^0_{(i-1),x[\mathsf{inp}'(i)]} & 0 \\ 0 & B^1_{(i-1),x[\mathsf{inp}'(i)]} \end{pmatrix} \\ &= \prod_{i=1}^{m} \begin{pmatrix} B^0_{i,x[\mathsf{inp}(i)]} & 0 \\ 0 & B^1_{i,x[\mathsf{inp}(i)]} \end{pmatrix} \\ &= \begin{pmatrix} \prod_{i=1}^m B^0_{i,x[\mathsf{inp}(i)]} & 0 \\ 0 & \prod_{i=1}^m B^1_{i,x[\mathsf{inp}(i)]} \end{pmatrix}\end{aligned}$$

Where the change of indices in the second step follows because $\mathsf{inp}'(i) = \mathsf{inp}(i-1)$ when $2 \le i \le m+1$. We now consider the two case for $b \in \{0,1\}$.

**Case 1: (b = 0)**

In this case,

$$\begin{aligned}\prod_{i=1}^{m+2} \widehat{B}^0_{i,x[\mathsf{inp}'(i)]} &= I_{2w\times 2w} \cdot \begin{pmatrix} \prod_{i=1}^m B^0_{i,x[\mathsf{inp}(i)]} & 0 \\ 0 & \prod_{i=1}^m B^1_{i,x[\mathsf{inp}(i)]} \end{pmatrix} \cdot I_{2w\times 2w} \\ &= \begin{pmatrix} \prod_{i=1}^m B^0_{i,x[\mathsf{inp}(i)]} & 0 \\ 0 & \prod_{i=1}^m B^1_{i,x[\mathsf{inp}(i)]} \end{pmatrix}\end{aligned}$$

as required.

**Case 2: (b = 1)**

In this case,

$$\begin{aligned}\prod_{i=1}^{m+2} \widehat{B}^1_{i,x[\mathsf{inp}'(i)]} &= \begin{pmatrix} 0 & I_{w\times w} \\ I_{w\times w} & 0 \end{pmatrix} \cdot \begin{pmatrix} \prod_{i=1}^m B^0_{i,x[\mathsf{inp}(i)]} & 0 \\ 0 & \prod_{i=1}^m B^1_{i,x[\mathsf{inp}(i)]} \end{pmatrix} \cdot \begin{pmatrix} 0 & I_{w\times w} \\ I_{w\times w} & 0 \end{pmatrix} \\ &= \begin{pmatrix} 0 & \prod_{i=1}^m B^1_{i,x[\mathsf{inp}(i)]} \\ \prod_{i=1}^m B^0_{i,x[\mathsf{inp}(i)]} & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & I_{w\times w} \\ I_{w\times w} & 0 \end{pmatrix} \\ &= \begin{pmatrix} \prod_{i=1}^m B^1_{i,x[\mathsf{inp}(i)]} & 0 \\ 0 & \prod_{i=1}^m B^0_{i,x[\mathsf{inp}(i)]} \end{pmatrix}\end{aligned}$$

as required. □

**Claim 5.** *For all $BP_0$ and $BP_1$ each of width $w$ and length $m$ on $n$ input bits, for each $b \in \{0,1\}$, for all $x \in \{0,1\}^n$,*

$$\mathsf{Merge}(BP_0, BP_1, b)(x) = BP_b(x)$$

*Proof.* Let $BP_0 = \{\mathsf{inp}(i), B^0_{i,0}, B^0_{i,1}\}_{i=1}^m$ and $BP_1 = \{\mathsf{inp}(i), B^1_{i,0}, B^1_{i,1}\}_{i=1}^m$. Define $\hat{BP}_0 = \mathsf{Merge}(BP_0, BP_1, 0)$ and $\hat{BP}_1 = \mathsf{Merge}(BP_0, BP_1, 1)$ as above. We observe that for any $x \in \{0,1\}^n$,

$$\begin{aligned}
&\mathsf{Merge}(BP_0, BP_1, b)(x) = 1 \\
\iff &(\prod_{i=1}^{m+2} \widehat{B}^b_{i,x[\mathsf{inp}'(i)]}) \cdot \mathbf{e}_1 = \mathbf{e}_1 \\
\iff &\begin{pmatrix} \prod_{i=1}^m B^b_{i,x[\mathsf{inp}(i)]} & 0 \\ 0 & \prod_{i=1}^m B^{1-b}_{i,x[\mathsf{inp}(i)]} \end{pmatrix} \cdot \mathbf{e}_1 = \mathbf{e}_1 \qquad \text{(from Claim 5)} \\
\iff &(\prod_{i=1}^{m} B^b_{i,x[\mathsf{inp}(i)]}) \cdot \mathbf{e}_1 = \mathbf{e}_1 \\
\iff &BP_b(x) = 1
\end{aligned}$$

Thus $\mathsf{Merge}(BP_0, BP_1, b)(x) = BP_b(x)$. □

The following claim illustrates some useful properties of the Merge procedure that we will use later. Firstly it notes that changing the bit Merge gets as input changes only the "switch" matrices in the first and last level of the program Merge outputs. Secondly, changing one level of a program Merge gets as input changes the output program in one level only. Finally, the first column of the output matrix of the widened program output by Merge depends only on the first column of the output matrix of the active program. The claim follows by observing the definition of Merge.

**Claim 6.** *Let $BP_0$ and$BP_1$ be length $m$, width $w$ branching programs, with input length $n$.*

- $\mathsf{Merge}(BP_0, BP_1, 0)$ *and* $\mathsf{Merge}(BP_0, BP_1, 1)$ *differ in only 4 matrices, the matrices corresponding to the first and last level.*
- *Let $BP_1'$ be a length $m$ branching program that differs from $BP_1$ in only the $i^{th}$ level for some $i \in [m]$. Then for both $b \in \{0,1\}$,* $\mathsf{Merge}(BP_0, BP_1, b)$ *and* $\mathsf{Merge}(BP_0, BP_1', b)$ *also differ only in the $i^{th}$ level. A similar statement holds for branching programs $BP_0'$ that differ from $BP_0$ in only one level.*
- *For any $b \in \{0,1\}$, let $BP = \mathsf{Merge}(BP_0, BP_1, b)$, and ${\mathsf{P_{out}}}^{BP}(\cdot)$ and ${\mathsf{P_{out}}}^{BP_b}(\cdot)$ be the functions computing the output matrices on a given input for $BP$ and $BP_b$ respectively. Then for every input $x \in \{0,1\}^n$,*

  $$\mathsf{col}_1({\mathsf{P_{out}}}^{BP}(x)) = \mathsf{extend}(\mathsf{col}_1({\mathsf{P_{out}}}^{BP_b}(x)))$$

  *where* extend *extends a length $w$ vector by appending $w$ zeroes to the end.*

## 4.2 Randomizing Branching Programs

We now describe Kilian's randomization technique [Kil88] for a branching program, adapted to our setting, by defining a process Rand that randomizes the matrices of a branching program $BP$. We will decompose the randomization into two parts, $\mathsf{Rand}^B$ and $\mathsf{Rand}^\alpha$, defined below, and define Rand as their composition.

**Definition 9** ($\mathsf{Rand}^B$)**.** *Let* $BP = \{\mathsf{inp}(i), B_{i,0}, B_{i,1}\}_{i=1}^m$ *be a branching program of width* $w$ *and length* $m$*, with length-*$n$ *inputs. Let* $p$ *be a prime exponential in* $n$*. Then the process* $\mathsf{Rand}^B(BP, p)$ *samples* $m$ *random invertible matrices* $R_1, R_2, \ldots, R_m \in Z_p^{w\times w}$ *uniformly and independently, and computes*

$$\tilde{B}_{i,b} = R_{(i-1)} \cdot B_{i,b} \cdot R_i^{-1} \quad \text{for every } i \in [m], \text{ and } b \in \{0,1\}$$

*where* $R_0$ *is defined as* $I_{w\times w}$*, and*

$$\mathbf{t} = R_m \cdot \mathbf{e}_1$$

$\mathsf{Rand}^B$ *then outputs*

$$(\{\tilde{B}_{i,b}\}_{i\in[m],b\in\{0,1\}}, \mathbf{t}, p)$$

**Definition 10** ($\mathsf{Rand}^\alpha$)**.** *Let* $(\{\tilde{B}_{i,b}\}_{i\in[m],b\in\{0,1\}}, \mathbf{t}, p)$ *be the output of* $\mathsf{Rand}^B(BP, p)$ *as defined above. On this input,* $\mathsf{Rand}^\alpha(\{\tilde{B}_{i,b}\}_{i\in[m],b\in\{0,1\}}, p)$ *samples* $2m$ *non-zero scalars* $\{\alpha_{i,b} \in \mathbb{Z}_p : i \in [m], b \in \{0,1\}\}$ *uniformly and independently, and outputs*

$$(\{\alpha_{i,b} \cdot \tilde{B}_{i,b}\}_{i\in[m],b\in\{0,1\}}, \mathbf{t})$$

**Definition 11** ($\mathsf{Rand}$)**.** *Let* $BP = \{\mathsf{inp}(i), B_{i,0}, B_{i,1}\}_{i=1}^m$ *be a branching program of width* $w$ *and length* $m$*, with length-*$n$ *inputs. Let* $p$ *be a prime exponential in* $n$*. Then we define* $\mathsf{Rand}(BP, p)$ *to be:*

$$\begin{aligned}\mathsf{Rand}(BP, p) &= (\mathsf{Rand}^\alpha(\mathsf{Rand}^B(BP, p))) \\ &= (\{\alpha_{i,b} \cdot \tilde{B}_{i,b}\}_{i\in[m],b\in\{0,1\}}, \mathbf{t})\end{aligned}$$

*Where* $(\{\alpha_{i,b} \cdot \tilde{B}_{i,b}\}_{i\in[m],b\in\{0,1\}}, \mathbf{t})$ *are as computed in the executions of* $\mathsf{Rand}^\alpha$ *and* $\mathsf{Rand}^B$*.*

**Execution of a randomized branching program:** To compute $BP(x)$ from the output of $\mathsf{Rand}(BP, p)$, given some input labelling function $\mathsf{inp} : [m] \to [n]$, and $x \in \{0,1\}^n$, we compute

$$\mathsf{Out}(x) = (\prod_{i=1}^m \alpha_{i,x[\mathsf{inp}(i)]} \cdot \tilde{B}_{i,x[\mathsf{inp}(i)]}) \cdot \mathbf{t}$$

Where $\mathsf{Out} \in Z_P^w$ is a $w \times 1$ vector. The intermediate multiplications cause each $R_i^{-1}$ to cancel each $R_i$, and $R_0 = I_{w\times w}$, so the above computation can also be expressed as:

$$\mathsf{Out}(x) = (\prod_{i=1}^m \alpha_{i,x[\mathsf{inp}(i)]} \cdot B_{i,x[\mathsf{inp}(i)]}) \cdot \mathbf{e}_1$$

When $BP(x) = 1$, we have that

$$\prod_{i=1}^m \alpha_{i,x[\mathsf{inp}(i)]} \cdot B_{i,x[\mathsf{inp}(i)]} \cdot \mathbf{e}_1 = (\prod_{i=1}^m \alpha_{i,x[\mathsf{inp}(i)]}) \cdot \mathbf{e}_1$$

When $BP(x) = 0$, we have that

$$\prod_{i=1}^m \alpha_{i,x[\mathsf{inp}(i)]} \cdot B_{i,x[\mathsf{inp}(i)]} \cdot \mathbf{e}_1 = (\prod_{i=1}^m \alpha_{i,x[\mathsf{inp}(i)]}) \cdot \mathbf{e}_k$$

for $k \neq 1$. Hence, to compute $BP(x)$, we compute $\mathsf{Out}(x)$ and output 0 if $\mathsf{Out}(x)[1] = 0$, and 1 otherwise.

**Simulating a randomized branching program:** Previous works ([BGK+13, BR13]) followed [Kil88] to show how to simulate the distribution of any single path corresponding to an input $x$ using just $BP(x)$. However, the simulator required that branching programs have unique accept and reject matrices $\mathsf{P}_{\mathsf{accept}}$ and $\mathsf{P}_{\mathsf{reject}}$.

We would also like a theorem, along the lines of [Kil88], that shows that any single path through a randomized branching program $BP$ corresponding to an input $x$ can be simulated knowing just the accept/reject behavior of $BP$ on $x$ (i.e. by knowing whether $BP(x) = 1$).

In our setting, however, branching programs only meet the relaxed requirement that the output matrix $\mathsf{P}_{\mathsf{out}}(x)$ computed by evaluating $BP$ on input $x$ satisfies $\mathsf{P}_{\mathsf{out}}(x) \cdot \mathbf{e}_1 = \mathbf{e}_1 \iff BP(x) = 1$. There can thus be multiple accept and reject matrices, and the particular accept or reject matrix output by $BP$ can depend both on $x$ and on the specific implementation of $BP$ (and not simply its accept/reject behavior). In contrast, in previous works, because $\mathsf{P}_{\mathsf{accept}}$ and $\mathsf{P}_{\mathsf{reject}}$ were unique, knowing just the accept/reject behavior of $BP$ on $x$ also determines $\mathsf{P}_{\mathsf{out}}(x)$.

What we will show is that, for the particular randomization scheme chosen above, we can simulate any single path through a randomized branching program $BP$ corresponding to an input $x$ without knowing the exact accept/reject matrix $\mathsf{P}_{\mathsf{out}}(x)$, but rather just knowing the first column $\mathsf{p}_{\mathsf{out}}(x) = \mathsf{col}_1(\mathsf{P}_{\mathsf{out}}(x))$.

This will be sufficient for our applications, because the class of branching programs we randomize will have the property that there are fixed columns $\mathsf{p}_{\mathsf{accept}}$ and $\mathsf{p}_{\mathsf{reject}} \in \mathbb{Z}_p^w$ such that for all $x \in \{0,1\}^n$, if $BP(x) = 1$ then $\mathsf{col}_1(\mathsf{P}_{\mathsf{out}}(x)) = \mathsf{p}_{\mathsf{accept}}$, and if $BP(x) = 0$ then $\mathsf{col}_1(\mathsf{P}_{\mathsf{out}}(x)) = \mathsf{p}_{\mathsf{reject}}$. In the case of such programs, $\mathsf{col}_1(\mathsf{P}_{\mathsf{out}}(x))$ is determined solely by $BP(x)$, and not the particular implementation of $BP$. Thus, for these programs, we can simulate given only $BP(x)$.

Before we show this theorem, we define notation for a path through a branching program corresponding to an input $x$.

**Definition 12** ($\mathsf{proj}_x$). *Let* $\mathsf{inp} : [m] \to [n]$ *be an input labelling function, and, for any* $x \in \{0,1\}^n$, *define* $\mathsf{proj}_x$, *relative to* $\mathsf{inp}$, *such that for any branching program* $BP$ *with labelling function* $\mathsf{inp}$, *for any prime* $p \in \mathbb{N}$, *and for any* $(\{\tilde{B}_{i,b}\}_{i\in[m],b\in\{0,1\}}, \mathbf{t}) \leftarrow \mathsf{Rand}^B(BP, p)$

$$\mathsf{proj}_x(\{\tilde{B}_{i,b}\}_{i\in[m],b\in\{0,1\}}, \mathbf{t}) = (\{\tilde{B}_{i,x[\mathsf{inp}(i)]}\}_{i\in[m]}, \mathbf{t}),$$

*that is,* $\mathsf{proj}_x$ *selects the elements from* $(\{\tilde{B}_{i,b}\}_{i\in[m],b\in\{0,1\}}, \mathbf{t})$ *used when evaluating input* $x$.

We now show a version of Kilian's theorem, adapted to our construction.

**Theorem 7.** *There exists an efficient simulator* $\mathsf{KSim}$ *such that the following holds. Let* $BP = \{\mathsf{inp}(i), B_{i,0}, B_{i,1}\}_{i\in[m]}$ *be a width-*$w$ *branching program of length* $m$ *on* $n$ *bit inputs, and* $p$ *a prime exponential in* $n$. *Let* $x \in \{0,1\}^n$ *be an input to* $BP$, *and let* $b_i = x[\mathsf{inp}(i)]$ *for each* $i \in [m]$. *Let* $\mathsf{P}_{\mathsf{out}}(x) = \prod_{i=1}^m B_{i,b_i}$ *denote the matrix obtained by evaluating* $BP$ *on* $x$, *and let* $\mathsf{p}_{\mathsf{out}}(x) = \mathsf{col}_1(\mathsf{P}_{\mathsf{out}}(x))$ *denote the first column of this output. Let* $\mathsf{proj}_x(\mathsf{Rand}^B(BP, p))$ *be defined respecting the labelling function* $\mathsf{inp}$. *Then* $\mathsf{KSim}(1^m, p, \mathsf{p}_{\mathsf{out}}(x))$ *is identically distributed to* $\mathsf{proj}_x(\mathsf{Rand}^B(BP, p))$.

*Proof.* We begin by defining $\mathsf{KSim}(1^n, p, BP(x))$ as follows:

- For each $i$, $\mathsf{KSim}$ selects $\tilde{B}_{i,b_i}$ to be a uniformly random invertible matrix in $Z_p^{w\times w}$.
- $\mathsf{KSim}$ selects $\mathbf{t} \in \mathbb{Z}_p^w$ solving

$$\Big(\prod_{i\in[m]} \tilde{B}_{i,b_i}\Big) \cdot \mathbf{t} = \mathsf{p}_{\mathsf{out}}(x) \qquad (1)$$

where $b_i = x[\mathsf{inp}(i)]$ for each $i$.

- $\mathsf{KSim}$ outputs $\{\{\tilde{B}_{i,b_i}\}_{i\in[m]}, \mathbf{t}\}$

We want to show that the distribution output by $\mathsf{KSim}$ matches the real distribution of $\{\{\tilde{B}_{i,b_i}\}_{i\in[m]}, \mathbf{t}\}$ in the output of $\mathsf{Rand}^B(BP, p)$. But from [Kil88], we have the following:

**Claim 8.** *The distribution of $\{\{\tilde{B}_{i,b_i}\}_{i\in[m]}, R_m\}$ can be exactly sampled given $\mathsf{P_{out}}(x)$, by sampling $\{\tilde{B}_{i,b_i}\}_{i\in[m]}$, $R_m$ to be uniformly random and independent invertible matrices in $\mathbb{Z}_p^{w\times w}$ subject to*

$$(\prod_{i\in[m]} \tilde{B}_{i,b_i}) \cdot R_m = \mathsf{P_{out}}(x) \tag{2}$$

The above claim implies the following:

**Claim 9.** *The distribution of $\{\{\tilde{B}_{i,b_i}\}_{i\in[m]}, R_m\}$ can be sampled by independently choosing each $\tilde{B}_{i,b_i}$ uniform and invertible, and fixing $R_m$ solving equation (2).*

*Proof.* This follows because for every choice of invertible $\tilde{B}_{i,b_i}$, there exists $R_m$ solving equation (2) given by

$$R_m = (\prod_{i\in[m]} \tilde{B}_{i,b_i}))^{-1} \cdot \mathsf{P_{out}}(x) \tag{3}$$

Further, every solution to equation (2) can be represented as invertible $\tilde{B}_{i,b_i}$, and an $R_m$ solving equation (3). Thus choosing a random solution to equation (2) corresponds to independently choosing each $\tilde{B}_{i,b_i}$ uniformly and invertible, and fixing $R_m$ solving equation (3). □

From the above argument, we have that the distribution of $\mathsf{proj}_x(\mathsf{Rand}(BP, p))$ is exactly the same as the distribution produced by independently choosing each $\tilde{B}_{i,b_i}$ uniform and invertible, fixing $R_m$ solving equation (3), setting $\mathbf{t}$ to be the first column of $R_m$, and outputting $\{\{\tilde{B}_{i,b_i}\}_{i\in[m]}, \mathbf{t}\}$. But note that each column $\mathsf{col}_i(R_m), i \in [w]$ is the unique solution to

$$(\prod_{i\in[m]} \tilde{B}_{i,b_i}) \cdot \mathsf{col}_i(R_m) = \mathsf{col}_i(\mathsf{P_{out}}(x))$$

Thus we have that each $\tilde{B}_{i,b_i}$ is independent, uniform, and invertible, and, using $i = 1$, $\mathbf{t}$ is the unique solution to

$$(\prod_{i\in[m]} \tilde{B}_{i,b_i}) \cdot \mathbf{t} = \mathsf{p_{out}}(x)$$

and, in particular, that $\mathbf{t}$ is determined by *only* the first column of $\mathsf{P_{out}}(x)$. Thus, we see that the distribution of $\mathsf{proj}_x(\mathsf{Rand}^B(BP, p))$ is exactly the same as that output by $\mathsf{KSim}$. □

### 4.3 Choosing a Set System

In this section we will describe how to choose a collection of sets under which to encode a randomized branching program using the graded encoding scheme. Our selection of sets will closely follow [BGK+13], in that we use straddling set systems. However, one difference is that while they use dual input branching programs, we restrict our attention to single-input schemes.

We first define straddling set systems.

**Definition 13** (Straddling Set Systems [BGK+13]). *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:*

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

*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:) $\bigcup_{S\in C} S = \bigcup_{S\in D} S$*

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

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

Informally, the guarantee provided by a straddling set system is that only way to exactly cover $U$ using elements from $\mathbb{S}_n$ is to use either all sets $\{S_{i,0}\}_{i\in n}$ or all sets $\{S_{i,1}\}_{i\in n}$. [BGK+13] give a construction for straddling set systems, choosing $U$ to be $[2n-1]$, each $S_{i,0}$ to be one of $\{1\}, \{2,3\}, \ldots, \{2n-2, 2n-1\}$, and each $S_{i,1}$ to be one of $\{1,2\}, \{3,4\}, \ldots, \{2n-1\}$. They further show that this construction is a straddling set system.

**Theorem 10** (From Construction 1 in [BGK+13]). *For every $n \in N$, there exists a straddling set system $\mathbb{S}_n$ with $n$ entries, over a universe $U$ of $2n-1$ elements.*

We now define the process SetSystem which takes as input the length $m$ of a branching program, the number of input bits $n$, and the input labelling function $\mathsf{inp} : [m] \to [n]$ for a branching program. SetSystem will output the collection of straddling set systems that we will use to encode any branching program of length $m$ on $n$ input bits, with labelling function inp.

**Execution of SetSystem$(m, n, \mathsf{inp})$:**
We let $n_j$ denote the number of levels that inspect the $j$th input bit in inp. That is,

$$n_j = |\{i \in [m] : \mathsf{inp}(i) = j\}|$$

For every $j \in [n]$, SetSystem chooses $\mathbb{S}^j$ to be a straddling set system with $n_j$ entries over a set $U_j$, such that the sets $U_1, \ldots, U_n$ are disjoint. Let $U = \bigcup_{j\in[n]} U_j$. SetSystem then chooses $S_t$ be a set disjoint from $U$. We associate the set system $\mathbb{S}^j$ with the $j$'th input bit of the branching program corresponding to inp. SetSystem then re-indexes the elements of $\mathbb{S}^j$ to match the steps of the branching program as described by inp, so that:

$$\mathbb{S}^j = \{S_{i,b} : \mathsf{inp}(i) = j, b \in \{0,1\}\}$$

By this indexing, we also have that $S_{i,b} \in \mathbb{S}^{\mathsf{inp}(i)}$ for every $i \in [m]$, for every $b \in \{0,1\}$.

Let $k = |U \cup S_t|$, and WLOG, assume that the $U^j$s and $S_t$ are disjoint subsets of $[k]$ (otherwise SetSystem relabels the elements to satisfy this property).

SetSystem then outputs

$$k, \quad \{S_{i,b}\}_{i\in[m], b\in\{0,1\}}, \quad S_t$$

### 4.4 Obfuscating Branching Programs

In this section, we will describe a process Obf that obfuscates a given branching program $BP$. This process will use Rand and SetSystem as subroutines. The output of Obf will be a randomized width-10 oblivious matrix branching program, encoded under the graded encoding scheme.

**Description of Obf$(BP)$ :**

**Input.** $\mathsf{Obf}$ takes as input an oblivious permutation branching program $BP = \{\mathsf{inp}(i), B_{i,0}, B_{i,1}\}_{i=1}^m$ of width $w$ and length $m$ on $n$ input bits.

**Choosing sets.** $\mathsf{Obf}$ runs $\mathsf{SetSystem}(m, n, \mathsf{inp})$ and receives $k, \{S_{i,b}\}_{i\in[m+2],b\in\{0,1\}}, S_t$.

**Initializing the GES.** $\mathsf{Obf}$ runs $\mathsf{InstGen}(1^n, 1^k)$, and receives secret parameters $\mathsf{sp}$ and public parameters $\mathsf{pp}$ which describe a $(k, R)$-graded encoding scheme. We assume the ring $R$ is equal to $\mathbb{Z}_p$ for some $p$ exponential in $n$.

**Randomizing BP.** $\mathsf{Obf}$ executes $\mathsf{Rand}(BP, p)$, and obtains its output, $\{\{\mathsf{inp}(i), \alpha_{i,0}\cdot\tilde{B}_{i,0}, \alpha_{i,1}\cdot\tilde{B}_{i,1}\}_{i\in[m]}, \mathbf{t}\}$

**Output.** $\mathsf{Obf}$ outputs:

$$\mathsf{pp}, \quad \{\mathsf{inp}(i), \quad \mathsf{Enc}(\mathsf{sp}, \alpha_{i,0} \cdot \tilde{B}_{i,0}, S_{i,0}), \quad \mathsf{Enc}(\mathsf{sp}, \alpha_{i,0} \cdot \tilde{B}_{i,0}, S_{i,1})\}_{i\in[m]}, \quad \mathsf{Enc}(\mathsf{sp}, \mathbf{t}, S_t)$$

We also define a generic version of $\mathsf{Obf}$, which we refer to as $\mathsf{GObf}$. It's output will be used to initialize an oracle $\mathcal{M}$ for the idealized version of the graded encoded scheme. $\mathsf{GObf}(BP)$ acts exactly as $\mathsf{Obf}(BP)$, except in the **Output** step, $\mathsf{GObf}$ outputs

$$\mathsf{pp}, \quad \{\mathsf{inp}(i), (\alpha_{i,0} \cdot \tilde{B}_{i,0}, S_{i,0}), (\alpha_{i,1} \cdot \tilde{B}_{i,1}, S_{i,1})\}_{i\in[m]}, \quad (\mathbf{t}, S_t)$$

### 4.5 Putting it all together: Obfuscating $\mathsf{NC}^1$ circuits

We now define our indistinguishability obfuscator $i\mathcal{O}$ for $\mathsf{NC}^1$, as follows:

**Description of $i\mathcal{O}(C)$ :**

1. $i\mathcal{O}$ takes as input $C \in \mathsf{NC}^1$, a fan-in 2 circuit with depth $d$ on $n$ input bits. $i\mathcal{O}$ uses Barrington's Theorem to convert $C$ into an oblivious width 5 permutation branching program $BP = \{\mathsf{inp}(i), B_{i,0}, B_{i,1}\}_{i=1}^m$ of length $m = 4^d$ on $n$ input bits.
2. $i\mathcal{O}$ generates a dummy width-5 branching program $I = \{\mathsf{inp}(i), I_{5\times5}, I_{5\times5}\}_{i=1}^m$ of length $m$, where each permutation matrix at each level is the identity matrix. $i\mathcal{O}$ then computes $\hat{BP} = \mathsf{Merge}(BP, I, 0)$.
3. $i\mathcal{O}$ outputs $\mathsf{Obf}(\hat{BP})$, which yields the public parameter $\mathsf{pp}$ for the graded encoding scheme, together with the encoded branching program $\{\mathsf{inp}(i), \mathsf{Enc}(\alpha_{i,0}\cdot\tilde{B}_{i,0}, S_{i,0}), \mathsf{Enc}(\alpha_{i,1}\cdot\tilde{B}_{i,1}, S_{i,1})\}_{i\in[m+2]}, \mathsf{Enc}(\mathbf{t}, S_t)$.

**Correctness of $i\mathcal{O}$:** In order to compute the output of $C(x)$ given its obfuscation $i\mathcal{O}(C)$, we perform matrix multiplication on the encoded matrices using the functions $\mathsf{Add}$ and $\mathsf{Mult}$ of the graded encoding scheme. That is, letting $b_i = x[\mathsf{inp}(i)]$ for each $i \in [m+2]$, using the $\mathsf{Eval}$ function guaranteed by Lemma 3, we compute the encoding of

$$\mathsf{Out}(x) = (\prod_{i=1}^{m+2} \alpha_{i,b_i} \cdot \tilde{B}_{i,b_i}) \cdot \mathbf{t}$$

and perform $\mathsf{isZero}$ on the encoding of $\mathsf{Out}(x)[1]$ (Note we can only apply $\mathsf{Eval}$ and Lemma 3 if the above computation is $\vec{S}$-respecting, but we will show that it is momentarily). From the correctness of the underlying randomized branching program, we have that $C(x) = 0 \iff \mathsf{Out}(x)[1] = 0$. Thus, $i\mathcal{O}$ is correct as long as the above computation is a $\vec{S}$-respecting circuit.

Note that when multiplying two matrices $M_1$ and $M_2$ encoded under $S_1$ and $S_2$ respectively, the multiplication is $\vec{S}$-respecting as long as $S_1 \cap S_2 = \emptyset$. Thus is suffices to show that the sets encoding the matrices being multiplied, namely:

$$S_{1,b_1}, \quad S_{2,b_2}, \quad \ldots, \quad S_{m+2,b_{m+2}}, \quad S_t$$

are all disjoint, and that their union is $[k]$.

Disjointness follows by observing that each of $U_1, U_2, \ldots, U_n, B_t$ is disjoint, and further that for each $j \in [n]$, for any $i, i'$ such that $\mathsf{inp}(i) = \mathsf{inp}(i') = j$, we have that $b_i = b_{i'} = x[\mathsf{inp}(i)]$ and $S_{i,b_i}$ and $S_{i',b_{i'}}$ are both elements of the straddling set system $\mathbb{S}^{\mathsf{inp}(i)}$, so $S_{i,b_i} \cap S_{i',b_{i'}} = \emptyset$.

To show that the union of the sets is $[k]$, we note that

$$(\bigcup_{i=1}^{m+2} S_{i,b_i}) \cup S_t = (\bigcup_{j=1}^{n} \bigcup_{i:\mathsf{inp}(i)=j} S_{i,x[j]}) \cup S_t = (\bigcup_{j=1}^{n} U_j) \cup S_t = [k]$$

by construction. Thus we have that $i\mathcal{O}$ is correct.

# 5 Proof of Indistinguishability Obfuscation

**Theorem 11.** *Assume the existence of semantically-secure multilinear encoding schemes. Then there exists indistinguishability obfuscators the the class of $\mathsf{NC}^1$ circuits.*

*Proof.* We show that the obfuscator defined in Section 4.5 is an indistinguishability obfuscator for $\mathsf{NC}^1$ circuits. Consider two $\mathsf{NC}^1$ circuit ensembles $\{C_n^0\}_{n\in\mathbb{N}}$ and $\{C_n^1\}_{n\in\mathbb{N}}$ such that for all $n \in \mathbb{N}$ and $x \in \{0,1\}^n$, $C_n^0(x) = C_n^1(x)$. Assume for contradiction there exists a nuPPT distinguisher $D$ and polynomial $p$ such that for infinitely many $n$, $D$ distinguishes $i\mathcal{O}(C_n^0)$ and $i\mathcal{O}(C_n^1)$ with advantage $1/p(n)$. For any $n \in \mathbb{N}$ let $BP_0$ and $BP_1$ be the branching programs of length $m = poly(n)$ obtained by applying Theorem 4 to the circuits $C_n^0$ and $C_n^1$ respectively.

We organize the proof in three parts. In the first part we show that if $D$ distinguishes between obfuscations of $C_n^0$ and $C_n^1$ then there exists widened branching programs $BP$ and $BP'$ that differ in only few matrices and evaluate the same function such that $D$ distinguishes between $\mathsf{Obf}(BP)$ and $\mathsf{Obf}(BP')$. Furthermore, the first column of the output matrix is the same for $BP$ and $BP'$, and depends only on the output of the program $BP(x) = BP'(x)$. More concretely, there exist vectors $v_0$ and $v_1$ such that for all inputs $x$ the first column of the output matrix for both $BP$ and $BP'$ is always $v_{BP(x)}$.

In the second part, we apply the semantic security of the graded encoding scheme used to argue that if $D$ distinguishes $\mathsf{Obf}(BP)$ and $\mathsf{Obf}(BP')$ then there exists an algebraic adversary that does the same. In particular, this adversary can distinguish between the oracles $\mathcal{M}(\mathsf{GObf}(BP))$ and $\mathcal{M}(\mathsf{GObf}(BP'))$. Finally, in the third part we show that these oracles can be simulated given oracle access to $BP$ (resp. $BP'$) and input $v_0$ and $v_1$. This, together with the fact that $BP$ and $BP'$ agree on all inputs, will imply a contradiction and hence prove the theorem.

## 5.1 Setting up $BP$ and $BP'$ via a hybrid argument

Let $\mathsf{Hyb}_i$ be a procedure that takes an input two length $m$ branching programs $P_0$ and $P_1$ (with the same labeling function) and outputs a "hybrid" length $m$ branching program whose first $i$ levels are identical to the first $i$ levels of $P_0$ and all the other levels are identical to those of $P_1$. Formally, let $P_0 = \{\mathsf{inp}(j), B_{j,0}, B_{j,1}\}_{j\in[m]}$ and $P_1 = \{\mathsf{inp}(j), B'_{j,0}, B'_{j,1}\}_{j\in[m]}$.

$$\mathsf{Hyb}_i(P_0, P_1) = \{\mathsf{inp}(j), B_{j,0}, B_{j,1}\}_{j=1}^{i}, \{\mathsf{inp}(j), B'_{j,0}, B'_{j,1}\}_{j=i+1}^{m}$$

For every $n \in \mathbb{N}$ we define hybrid distributions in the following way.

- We start with $H_0$ which is the obfuscation of the circuit $C_n^0$.

$$H_0 = i\mathcal{O}(C_n^0) = \mathsf{Obf}(\mathsf{Merge}(BP_0, I, 0))$$

- For $i = 1, 2 \dots m$, let

$$H_i = \mathsf{Obf}(\mathsf{Merge}(BP_0, \mathsf{Hyb}_i(BP_1, I), 0))$$

We change, one level at a time, the second branching program $\mathsf{Merge}$ takes as input from $I$ to $BP_1$.

- We have that $H_m = \mathsf{Obf}(\mathsf{Merge}(BP_0, BP_1, 0))$. We change the "switch" input to $\mathsf{Merge}$ so that the second branching program $BP_1$ is active.

$$H_{m+1} = \mathsf{Obf}(\mathsf{Merge}(BP_0, BP_1, 1))$$

- For $i = 1, 2 \dots m$, let

$$H_{m+i+1} = \mathsf{Obf}(\mathsf{Merge}(\mathsf{Hyb}_i(BP_1, BP_0), BP_1, 1))$$

We change the first program $\mathsf{Merge}$ takes as input from $BP_0$ to $BP_1$, one level at a time as before.

- We have that $H_{2m+1} = \mathsf{Obf}(\mathsf{Merge}(BP_1, BP_1, 1))$. We switch back so that the first program is active (which in this case is the same as the second program $BP_1$)

$$H_{2m+2} = \mathsf{Obf}(\mathsf{Merge}(BP_1, BP_1, 0))$$

- For $i = 1, 2 \dots m$, let

$$H_{2m+i+2} = \mathsf{Obf}(\mathsf{Merge}(BP_1, \mathsf{Hyb}_i(I, BP_1), 0))$$

We change the second program $\mathsf{Merge}$ takes as input from $BP_1$ to $I$, one level at a time as before. Finally we get

$$H_{3m+2} = i\mathcal{O}(C_n^1) = \mathsf{Obf}(\mathsf{Merge}(BP_1, I, 0))$$

which is the obfuscation of the circuit $C_n^1$.

Recall that by assumption $D$ distinguishes between $\{i\mathcal{O}(C_n^0)\}_{n\in\mathbb{N}}$ and $\{i\mathcal{O}(C_n^1)\}_{n\in\mathbb{N}}$. That is, there is a polynomial $p$ such that for infinitely many $n$

$$|Pr[D(H_0) = 1] - Pr[D(H_{3m+2})]| > 1/p(n)$$

By the above hybrid argument, $D$ must distinguish between a pair of consecutive hybrids. That is, there exists some $i \in \{0, 1, \dots 3m + 1\}$ such that

$$|Pr[D(H_i) = 1] - Pr[D(H_{i+1})]| > 1/4mp(n)$$

We now show that $H_i$ and $H_{i+1}$ can be expressed as the $\mathsf{Obf}(BP)$ and $\mathsf{Obf}(BP')$ respectively where $BP$ and $BP'$ are (widened) branching programs that differ in only two levels and agree on all inputs. Furthermore, both $BP$ and $BP'$ have the property that for all inputs $x$ the first column of the output matrix $\mathsf{col}_1(\mathsf{P}_{\mathsf{out}}(x))$ is the same for $BP$ and $BP'$, and depends only on the output of these programs on $x$. More formally,

**Claim 12.** *There exist branching programs $BP$ and $BP'$ of length $m' = m + 2$ and width 10 such that*

- $H_i = \mathsf{Obf}(BP)$ *and* $H_{i+1} = \mathsf{Obf}(BP')$.

- *$BP$ and $BP'$ differ in at most 2 levels.*
- *For all $x \in \{0,1\}^n$, $BP(x) = BP'(x)$.*
- *Let $\mathsf{P_{out}}^{BP}(\cdot)$ and $\mathsf{P_{out}}^{BP'}(\cdot)$ be the functions computing the output matrices for $BP$ and $BP'$ respectively. There exist length 10 vectors $v_0$ and $v_1$ such that for every $x \in \{0,1\}^n$, $\mathsf{col}_1(\mathsf{P_{out}}^{BP}(x)) = \mathsf{col}_1(\mathsf{P_{out}}^{BP'}(x)) = v_{BP(x)}$*

*Proof.* Let $v_1 = \mathsf{extend}(\mathsf{col}_1(\mathsf{P_{accept}}))$ and $v_0 = \mathsf{extend}(\mathsf{col}_1(\mathsf{P_{reject}}))$ where $\mathsf{P_{accept}}$ and $\mathsf{P_{reject}}$ are the accepting and rejecting matrices from Theorem 4 for branching programs of input lengths $n$, and $\mathsf{extend}$ extends a length $w$ vector by appending $w$ zeroes. We consider three cases: when $i$ is equal to $m$, $2m+1$ and otherwise.

**Case 1: $i = m$:** By definition of $H_i$ and $H_{i+1}$, the branching programs $BP$ and $BP'$ are $\mathsf{Merge}(BP_0, BP_1, 0)$ and $\mathsf{Merge}(BP_0, BP_1, 1)$ respectively. By Claim 7, $BP$ and $BP'$ differ in the "switch" matrices, which make up the first and last level. Furthermore, $BP$ and $BP'$ compute $BP_0$ and $BP_1$ respectively which are equivalent programs by assumption. It remains to show the fourth condition. By Claim 7, the first column of the output matrix for a widened branching program only depends on the first column of the output matrix of the active program. Hence, for every input $x$, $\mathsf{col}_1(\mathsf{P_{out}}^{BP}(x)) = \mathsf{extend}(\mathsf{col}_1(\mathsf{P_{out}}^{BP_0}(x)))$. By Theorem 4, $\mathsf{P_{out}}^{BP_0}(x)$ is either $\mathsf{P_{accept}}$ or $\mathsf{P_{reject}}$ depending on the output $BP_0(x)$. Therefore, for all inputs $x$ such that $BP(x) = 0$,

$$\mathsf{col}_1(\mathsf{P_{out}}^{BP}(x)) = \mathsf{extend}(\mathsf{col}_1(\mathsf{P_{reject}})) = v_0$$

Similarly, for all inputs $x$ such that $BP(x) = 1$,

$$\mathsf{col}_1(\mathsf{P_{out}}^{BP}(x)) = \mathsf{extend}(\mathsf{col}_1(\mathsf{P_{accept}})) = v_1$$

The same argument holds for $BP'$ too, in which case $BP_1$ is active and has the same accepting and rejecting permutations $\mathsf{P_{accept}}$ and $\mathsf{P_{reject}}$ by Theorem 4. Therefore, for all inputs $x$,

$$\mathsf{col}_1(\mathsf{P_{out}}^{BP'}(x)) = v_{BP_1(x)}$$

Since $BP_0(x) = BP_1(x) = BP(x)$ for all $x$, the claim follows.

**Case 2: $i = 2m+1$:** By definition of $H_i$ and $H_{i+1}$, the branching programs $BP$ and $BP'$ are $\mathsf{Merge}(BP_1, BP_1, 0)$ and $\mathsf{Merge}(BP_1, BP_1, 1)$ respectively. As before, these programs differ in the first and level only. Furthermore, both $BP$ and $BP'$ compute the same function, as the active program is the same ($BP_1$). Also, directly from Claim 7 and Theorem 4 we have that for all inputs $x \in \{0,1\}^n$,

$$\mathsf{col}_1(\mathsf{P_{out}}^{BP}(x)) = \mathsf{col}_1(\mathsf{P_{out}}^{BP'}(x)) = \mathsf{extend}(\mathsf{col}_1(\mathsf{P_{out}}^{BP_1}(x))) = v_{BP_1(x)} = v_{BP(x)}$$

**Case 3: $i \neq m$ and $i \neq 2m+1$:** First, consider the subcase when $i < m$ or $i > 2m+1$. The programs $BP$ and $BP'$ are of the form $\mathsf{Merge}(BP_0, P_i)$ and $\mathsf{Merge}(BP_0, P_{i+1})$ respectively where $P_i$ and $P_{i+1}$ are branching programs that differ only in the $i+1^{th}$ level. By Claim 7, $BP$ and $BP'$ differ only in the $i+1^{th}$ level too. Furthermore, in both $BP$ and $BP'$, the active program is $BP_0$. Hence $BP$ and $BP'$ compute the same function and similarly as the previous case, we have that for all inputs $x \in \{0,1\}^n$,

$$\mathsf{col}_1(\mathsf{P_{out}}^{BP}(x)) = \mathsf{col}_1(\mathsf{P_{out}}^{BP'}(x)) = \mathsf{extend}(\mathsf{col}_1(\mathsf{P_{out}}^{BP_0}(x))) = v_{BP_0(x)} = v_{BP(x)}$$

The case when $m < i < 2m+1$ follows similarly. This concludes the proof of the claim. □

This concludes the first part of the proof. At this point we have that there is a polynomial $p$ such that for infinitely many $n$ there exist branching programs $BP$ and $BP'$ with the properties described in Claim 13 such that

$$|Pr[D(\mathsf{Obf}(BP)) = 1] - Pr[D(\mathsf{Obf}(BP'))]| > 1/4mp(n)$$

In the next part we show that the distinguisher $D$ can be used to break the semantic security game of the graded encoding scheme used by $\mathsf{Obf}$.

## 5.2 Applying Semantic Security

Fix $n \in \mathbb{N}$, and let $BP = \{\mathsf{inp}(i), B_{i,0}, B_{i,1}\}_{i\in[m']}$ and $BP' = \{\mathsf{inp}(i), B'_{i,0}, B'_{i,1}\}_{i\in[m']}$. Let $l_1, l_2 \in [m]$ be the levels in which $BP$ and $BP'$ differ. All other matrices of $BP$ and $BP'$ are the same. That is, for every $i \notin \{l_1, l_2\}, b \in \{0,1\}$ we have that $B_{i,b} = B'_{i,b}$.

Define nuPPT $M$ that gets $BP$ and $BP'$ as non-uniform advice and on input the public parameters $\mathsf{pp}$ that describe a $(\mathbb{Z}_p, k)$-graded encoding scheme samples $m'$ random invertible $10 \times 10$ matrices over $\mathbb{Z}_p$, $\{R_i\}_{i\in[m']}$ and $2m'$ random scalars from $\mathbb{Z}_p$, $\{\alpha_{i,b}\}_{i\in[m'],b\in\{0,1\}}$. $M$ then uses these matrices and scalars to randomize $BP$ and $BP'$ as described by $\mathsf{Rand}(\cdot, p)$ to obtain $\{\alpha_{i,b} \cdot \tilde{B}_{i,b}\}_{i\in[m'],b\in\{0,1\}}$, $\{\alpha_{i,b} \cdot \tilde{B}'_{i,b}\}_{i\in[m'],b\in\{0,1\}}$ and $\mathbf{t}$. $M$ outputs

$$\vec{m_0} = (\{\alpha_{l_1,b} \cdot \tilde{B}_{l_1,b}\}_{b\in\{0,1\}}, \{\alpha_{l_2,b} \cdot \tilde{B}_{l_2,b}\}_{b\in\{0,1\}})$$

$$\vec{m_1} = (\{\alpha_{l_1,b} \cdot \tilde{B}'_{l_1,b}\}_{b\in\{0,1\}}, \{\alpha_{l_2,b} \cdot \tilde{B}'_{l_2,b}\}_{b\in\{0,1\}})$$

$$\vec{z} = (\{\alpha_{i,b} \cdot \tilde{B}_{i,b}\}_{i\in[m']/\{l_1,l_2\},b\in\{0,1\}}, \mathbf{t})$$

We observe that $D(\mathsf{Obf}(BP))$ (resp. $D(\mathsf{Obf}(BP'))$) is simply the output of $D$ when playing the semantic security game with the message sampler $M$ and parameterized by the bit $b = 0$ (resp. $b = 1$). Formally, there exist polynomials $q, k$ constant $c$ and set ensembles $\{(\vec{S}_n, \vec{T}_n)\}_{n\in\mathbb{N}}$ such that for all $n \in \mathbb{N}$

$$D(\mathsf{Obf}(BP)) \equiv \mathbf{Output}_0(q, k, c, D, M, n, (\vec{S}_n, \vec{T}_n))$$

and

$$D(\mathsf{Obf}(BP')) \equiv \mathbf{Output}_1(q, k, c, D, M, n, (\vec{S}_n, \vec{T}_n))$$

where $\vec{S}_n$, $\vec{T}_n$ contain sets from $\mathsf{SetSystem}(m', n, \mathsf{inp})$ and $\mathbf{Output}_b$ is as defined in Definition 9.

To see this, observe that $\vec{m_0}$ and $\vec{m_1}$ consist of a constant number of ring elements while $\vec{z}$ contains polynomially many ring elements. Note that the distribution of $(\vec{m_0}, \vec{z})$ is identical to $\mathsf{Rand}(BP, p)$ and the distribution of $(\vec{m_1}, \vec{z})$ is identical to $\mathsf{Rand}(BP', p)$. When these elements are encoded under sets in $\mathsf{SetSystem}(m', n, \mathsf{inp})$ [8] then we obtain the distributions $\mathsf{Obf}(BP)$ and $\mathsf{Obf}(BP')$ respectively.

Recall that for infinitely many $n$,

$$|Pr[D(\mathsf{Obf}(BP)) = 1] - Pr[D(\mathsf{Obf}(BP'))]| > 1/4mp(n)$$

Since the graded encoding scheme is semantically secure, it must be that $M$ is not a $(q, k, c, \{(\vec{S}_n, \vec{T}_n)\}_{n\in\mathbb{N}})$-respecting message sampler. Therefore, there exists a polynomial $p'$ and algebraic adversary nuPPT $A$ such that for infinitely many $n \in \mathbb{N}$,

$$Pr[A^{\mathcal{M}(\mathsf{GObf}(BP))}(1^n) = 1] - Pr[A^{\mathcal{M}(\mathsf{GObf}(BP'))}(1^n) = 1] > 1/p'(n)$$

In the remainder of the proof we show that if $BP$ and $BP'$ agree on all inputs then such algebraic adversary $A$ cannot exist. More precisely, we will show that the oracles $\mathcal{M}(\mathsf{GObf}(BP))$ and $\mathcal{M}(\mathsf{GObf}(BP'))$ can be simulated with only oracle access to $BP$ and $BP'$.

Similar statements were shown in [BGK+13] and [BR13]. In particular, $\mathsf{GObf}$ is a simplified version of the obfuscator of [BGK+13], which [BGK+13] shows is VBB secure against algebraic adversaries. We will follow the structure of the proof in [BGK+13], but cannot use it in a black-box way due to the differences in the construction and the fact that their proof only works for branching programs that have unique accepting and rejecting output matrices. The branching programs we consider may not have this property.

[8] Every element is encoded with the corresponding set in $\mathsf{SetSystem}(m', n, \mathsf{inp})$. For example, elements from $\alpha_{i,b}\tilde{B}_{i,b}$ are encoded under the set $S_{i,b}$ in $\mathsf{SetSystem}(m', n, \mathsf{inp})$

### 5.3 Simulating an algebraic adversary

We show the following claim.

**Claim 13.** *There exists Turing machine* Sim *such that for every nuPPT $A$ there exists a negligible function $\varepsilon$ such that the following holds. For every $n \in \mathbb{N}$, every length $m = poly(n)$ and width 10 branching program $BP$ with labeling function* $\mathsf{inp} : [m] \to [n]$ *and for which there exist $v_0, v_1 \in \{0,1\}^{10}$ such that on every input $x \in \{0,1\}^n$, $\mathsf{col}_1(\mathsf{P}_{\mathsf{out}}(x)) = v_{BP(x)}$, it holds that*

$$\Delta(A^{\mathcal{M}(\mathsf{GObf}(BP))}(1^n), \mathsf{Sim}^{BP}(1^n, 1^m, A, \mathsf{inp}, v_0, v_1)) \le \varepsilon(n)$$

Sim's strategy will be to run $A$ and simulate the oracle $\mathcal{M}(\mathsf{GObf}(BP))$ for $A$. Recall that $\mathsf{GObf}(BP)$ contains the public parameters of the encoding scheme pp and a list of the ring elements in $\mathsf{Rand}(BP)$ paired with the corresponding set in $\vec{S} = \mathsf{SetSystem}(n, m, \mathsf{inp})$. $\mathcal{M}(\mathsf{GObf}(BP))$ when queried with an arithmetic circuit $C$, first checks if $C$ is $\vec{S}$-respecting and then outputs the result of $C$ on the ring elements in $\mathsf{Rand}(BP)$.

Sim only has oracle access to $BP$ and can not run $\mathcal{M}$ directly on $\mathsf{GObf}(BP)$. However, we show in the following lemma that Sim can simulate the output of $\mathcal{M}$ on a single query. In particular, except with negligible probability (over $\mathsf{GObf}(BP)$ and the simulation) the simulated query response will be *identical* to the actual query response. Since $A$ makes only polynomially many queries, by the Union Bound, it follows that except with negligible probability Sim succeeds in correctly simulating all queries. Therefore, it suffices to show the following lemma.

**Lemma 14.** *There exists a Turing machine* CSim *such that for every $m, n, w \in \mathbb{N}$, $v_0, v_1 \in \{0,1\}^w$, labeling function* $\mathsf{inp} : [m] \to [n]$*, prime number $p$, and $\vec{S}$-respecting arithmetic circuit $C$ where $\vec{S} =$* $\mathsf{SetSystem}(m, n, \mathsf{inp})$*, the following holds. For every branching program $BP$ of length $m$, width $w$ and labeling function* inp *for which on every input $x$, $\mathsf{col}_1(\mathsf{P}_{\mathsf{out}}(x)) = v_{BP(x)}$ it holds that*

$$Pr[\mathsf{isZero}(C(\mathsf{Rand}(BP, p))) \ne \mathsf{CSim}^{BP}(1^m, p, C, v_0, v_1)] \le 32wm/p$$

The proof of the lemma follows the structure of the VBB simulation in [BGK+13], appropriately adapted to deal with the fact that our branching programs do not have a unique output by relying on Theorem 8.

*Proof.* Roughly speaking the lemma follows from the the property that $\vec{S}$-respecting arithmetic circuits, due to the straddling set systems in $\vec{S}$, can only evaluate expressions that are "consistent" with some inputs. In particular, following [BGK+13], the polynomial $C$ evaluates can be expressed as the sum of *single-input terms* where each *single-input term* is a function of those elements of that are consistent with some input to the branching program. Next, we rely on Theorem 8 to show that the sum of these single-input terms will depend only on the value of the branching program on these inputs.

The following proposition states that the function a $\vec{S}$-respecting arithmetic circuit computes can be expressed as the sum of several *single-input* terms. This decomposition is very similar to the one shown in [BGK+13].[9]

**Proposition 1.** *Fix $m, n, w \in \mathbb{N}$ and* $\mathsf{inp} : [m] \to [n]$*. Let $\vec{S} = \mathsf{SetSystem}(m, n, \mathsf{inp}) = (\{S_{i,b}\}_{i \in [m], b \in \{0,1\}}, S_t)$, and let $C$ be any $\vec{S}$-respecting arithmetic circuit. There exists a set $X \subseteq \{0,1\}^n$ of inputs such that*

**(i)**

$$C \equiv \sum_{x \in X} C_x$$

[9]The key difference is that [BGK+13] proves such a decomposition for "dual-input" branching program. We will later rely on a such a decomposition (for "dual-input" program) when constructing extractability obfuscators. Another minor difference is that since our scheme is slightly different, the terms of the decomposition are slightly different as well.

*where each $C_x$ is a $\vec{S}$-respecting arithmetic circuit, whose input wires are labelled only with sets respecting a single input $x \in \{0,1\}^n$, that is, only with sets $\in \{S_{i,x[\mathsf{inp}(i)]}\}_{i\in[m]} \cup \{S_t\}$.*

**(ii)** *For each $C_x$ above, for every branching program $BP$ of width $w$ and length $m$ on $n$ input bits, with input labelling function $\mathsf{inp}$, every prime $p$, and every $(\{\alpha_{i,b} \cdot \tilde{B}_{i,b}\}_{i\in m, b\in\{0,1\}}, \mathbf{t}) \leftarrow \mathsf{Rand}(BP, p)$*

$$C_x(\{\alpha_{i,b}\tilde{B}_{i,b}\}_{i\in[m],b\in\{0,1\}}, \mathbf{t}) = \alpha_x \cdot p_x(\{\tilde{B}_{i,x[\mathsf{inp}(i)]}\}_{i\in[m]}, \mathbf{t})$$

*where $p_x$ is some polynomial, and $\alpha_x = (\prod_{i\in[m]} \alpha_{i,x[\mathsf{inp}(i)]})$. Furthermore, when $p_x$ is viewed as a sum of monomials, each monomial contains exactly one entry from each $\tilde{B}_{i,x[\mathsf{inp}(i)]}$, and one entry from $\mathbf{t}$.*

The proof of Proposition 1 closely follows [BGK+13]; for completeness we provide a complete proof in Appendix B.

Now we are ready to describe the simulator $\mathsf{CSim}$. $\mathsf{CSim}$ gets as input $1^m$, prime $p$, a $\vec{S}$-respecting circuit $C$, vectors $v_0$, $v_1$ and has oracle access to a length $m$ branching program $BP$. Let $X$ be the set of inputs and $\{p_x\}_{x\in X}$ be the single-input polynomials corresponding to the decomposition of $C$. For every $x \in X$, $\mathsf{CSim}$ queries $BP$ on $x$, samples $d_x \leftarrow \mathsf{KSim}(1^m, p, v_{BP(x)})$ and checks whether $p_x(d_x) = 0$. $\mathsf{CSim}$ outputs 1 if and only if for every input $x \in X$, $p_x(d_x) = 0$.

Now we prove correctness of our simulation. First, we prove some claims that will be useful. In each of these claims, let $\mathsf{proj}_x$ be defined with respect to the labeling function $\mathsf{inp}$ of the branching program $BP$. The following claim states that if $C(\mathsf{Rand}(BP, p))$ is always zero, then every single-input term is always zero.

**Claim 15.** *If $Pr[C(\mathsf{Rand}(BP, p) = 0] = 1$ then for every input $x \in X$,*

$$Pr[p_x(\mathsf{proj}_x(\mathsf{Rand}^B(BP, p))) = 0] = 1$$

*Proof.* Consider a fixed $d = (\{\tilde{B}_{i,b}\}_{i\in[m],b\in\{0,1\}}, \mathbf{t})$ in the support of $\mathsf{Rand}^B(BP, p)$ and let $C_d(\{\alpha_{i,b}\}_{i\in[m],b\in\{0,1\}}) = C(\{\alpha_{i,b} \cdot \tilde{B}_{i,b}\}_{i\in[m],b\in\{0,1\}}, \mathbf{t})$. By Proposition 1, we know that

$$C_d(\{\alpha_{i,b}\}) = \sum_{x\in X}(\prod_{i\in[m]} \alpha_{i,x[\mathsf{inp}(i)]})p_x(\mathsf{proj}_x(d))$$

and $C_d$ is a degree $m + 2$ polynomial. By assumption, $C(\mathsf{Rand}(BP, p))$ is always zero (over the support of $\mathsf{Rand}(BP, p)$); hence, $C_d(\{\alpha_{i,b}\}) = 0$ for all non-zero $\{\alpha_{i,b}\}$. By the Schwartz-Zippel lemma, this can happen only if $C_d$ is the zero polynomial. By the structure of $C_d$, this implies that for every $x \in X$, $p_x(\mathsf{proj}_x(d)) = 0$. This argument works for every fixed value of $d$, hence we have that for every $x \in X$, $\Pr[p_x(\mathsf{proj}_x(\mathsf{Rand}^B(BP, p))) = 0] = 1$. □

The next claim states that if $C(\mathsf{Rand}(BP, p))$ is not always zero, then it is zero with small probability. Furthermore, there exists a single-input term that is zero with small probability.

**Claim 16.** *For any $\vec{S}$-respecting circuit $C$, if $Pr[C(\mathsf{Rand}(BP, p)) = 0] < 1$ then the following holds.*

1. *$Pr[C(\mathsf{Rand}(BP, p)) = 0] \leq 16wm/p$*
2. *There exists $x \in X$ such that $Pr[p_x(\mathsf{proj}_x(\mathsf{Rand}^B(BP, p))) = 0] \leq 16wm/p$, where $X$ is obtained from the decomposition of $C$ by Proposition 1.*

*Proof.* We start by showing part 1.

**Part 1:** If $\mathsf{Rand}(BP, p) = \mathsf{Rand}^\alpha(\mathsf{Rand}^B(BP, p))$ can be expressed as a low-degree ($\leq 2w$) polynomial on uniformly random values in $\mathbb{Z}_p$—namely, the $\alpha$'s and the randomization matrices $R_i$'s—then by the Schwartz-Zippel lemma the first part of the claim directly follows. However, there are two barriers to applying this argument:

- $\mathsf{Rand}^B$ does not sample uniformly random matrices $\{R_i\}_{i\in[m]}$; rather, it chooses uniformly random *invertible* matrices $R_i$. Similarly, $\mathsf{Rand}^\alpha$ does not sample uniformly random $\{\alpha_{i,b}\}_{i\in[m],b\in\{0,1\}}$; rather, it chooses uniformly random *non-zero* $\alpha_{i,b}$.
- $\mathsf{Rand}^B$ also needs to compute inverses $R_i^{-1}$ to $R_i$ for every $i \in [m]$ (which may no longer be expressed as low degree polynomials in the matrices $\{R_i\}_{i\in[m]}$).

To handle the second issue, consider the distribution $\mathsf{Rand}^B_{adj}(BP, p)$ that is defined exactly as $\mathsf{Rand}^B(BP, p)$ except that for every $i \in [m]$ it uses $adj(R_i) = R_i^{-1}det(R_i)$ instead of $R_i^{-1}$. Note that every entry of the adjoint of a $w \times w$ matrix $M$ is some cofactor of $M$ (obtained by the determinant of the $w-1 \times w-1$ matrix obtained by deleting some row and column of $A$). Hence every entry of $adj(R_i)$ can be expressed as a degree $w$ polynomial in $R_i$. Let $\mathsf{Rand}_{adj}(BP, p) = \mathsf{Rand}^\alpha(\mathsf{Rand}^B_{adj}(BP, p))$. It follows that $\mathsf{Rand}_{adj}(BP, p)$ is computed by degree (at most) $2w$ polynomial in the matrices $\{R_i\}_{i\in[m]}$ and scalars $\{\alpha_{i,b}\}_{i\in[m],b\in\{0,1\}}$.

Furthermore, we show that $Pr[C(\mathsf{Rand}_{adj}(BP, p)) = 0] = Pr[C(\mathsf{Rand}(BP, p)) = 0]$. Recall that by Proposition 1,

$$C \equiv \sum_{x\in X} C_x$$

and for each $C_x$ above and every $(\{\alpha_{i,b} \cdot \tilde{B}_{i,b}\}_{i\in[m],b\in\{0,1\}}, \mathbf{t}) \leftarrow \mathsf{Rand}(BP, p)$ ,

$$C_x(\{\alpha_{i,b} \cdot \tilde{B}_{i,b}\}_{i\in[m],b\in\{0,1\}}, \mathbf{t}) = \alpha_x \cdot p_x(\{\tilde{B}_{i,x[\mathsf{inp}(i)]}\}_{i\in[m]}, \mathbf{t})$$

where $\alpha_x = (\prod_{i\in[m]} \alpha_{i,x[\mathsf{inp}(i)]})$ and $p_x$ is a polynomial such that, when viewed as a sum of monomials, each monomial contains exactly one entry from each $\tilde{B}_{i,x[\mathsf{inp}(i)]}$, and one entry from $\mathbf{t}$. Recall that for every $i \in [m]$,

$$\tilde{B}_{i,x[\mathsf{inp}(i)]} = R_{i-1} B_{i,x[\mathsf{inp}(i)]} R_i^{-1}$$

For every $i \in [m]$, replacing $R_i^{-1}$ with $adj(R_i)$ has the effect of multiplying each monomial in $p_x$ with the scalar $det(R_i)$. Hence

$$C_x(\mathsf{Rand}_{adj}(BP, p)) = (\prod_{i\in[m]} det(R_i)) \cdot C_x(\mathsf{Rand}(BP, p))$$

Since $C$ is the sum of such $C_x$ terms, it holds that $C(\mathsf{Rand}_{adj}(BP, p)) = (\prod_{i\in[m]} det(R_i))C(\mathsf{Rand}(BP, p))$. For every $i \in [m]$, by invertibility, $det(R_i) \neq 0$ and hence

$$Pr[C(\mathsf{Rand}_{adj}(BP, p)) = 0] = Pr[C(\mathsf{Rand}(BP, p)) = 0]$$

So far, we have that $\mathsf{Rand}_{adj}(BP, p)$ is computed by a degree $2w$ polynomial in the matrices $\{R_i\}_{i\in[m]}$ and scalars $\{\alpha_{i,b}\}_{i\in[m],b\in\{0,1\}}$. However the first issue remains: each $R_i$ is uniformly random invertible and each $\alpha_{i,b}$ is uniformly random non-zero, whereas we need them to be uniformly random. Consider the distribution $\mathsf{Rand}_{adj,U}(BP, p)$ that is obtained by the computing the same polynomial on uniformly random matrices $\{R_i\}_{i\in[m]}$ and scalars $\{\alpha_{i,b}\}_{i\in[m],b\in\{0,1\}}$ over $\mathbb{Z}_p$. In Claim 23, we show that the statistical distance between $\mathsf{Rand}_{adj}(BP, p)$ and $\mathsf{Rand}_{adj,U}(BP, p)$ is at most $8wm/p$. Furthermore, the support of $\mathsf{Rand}_{adj,U}(BP, p)$ contains the support of $\mathsf{Rand}_{adj}(BP, p)$. This implies that if $Pr[C(\mathsf{Rand}_{adj}(BP, p)) = 0] < 1$ then $Pr[C(\mathsf{Rand}_{adj,U}(BP, p)) = 0] < 1$.

We now turn to proving the statement of the claim. Using facts shown above, we have that

$$Pr[C(\mathsf{Rand}(BP, p)) = 0] < 1 \implies Pr[C(\mathsf{Rand}_{adj}(BP, p)) = 0] < 1 \implies Pr[C(\mathsf{Rand}_{adj,U}(BP, p)) = 0] < 1$$

By Proposition 1, $C$ evaluates a $m+1$ degree polynomial, and $\mathsf{Rand}_{adj,U}(BP,p)$ is computed by a degree $2w$ polynomial in uniformly random values in $\mathbb{Z}_p$. By the Schwartz-Zippel lemma,

$$Pr[C(\mathsf{Rand}_{adj,U}(BP,p)) = 0] < 1 \implies Pr[C(\mathsf{Rand}_{adj,U}(BP,p) = 0] \leq 2w(m+1)/p \leq 8wm/p$$

We have that the statistical distance between $\mathsf{Rand}_{adj,U}(BP,p)$ and $\mathsf{Rand}_{adj}(BP,p)$ is at most $8wm/p$. Therefore, $Pr[C(\mathsf{Rand}(BP,p)) = 0] = Pr[C(\mathsf{Rand}_{adj}(BP,p)) = 0] \leq 16wm/p$ thus proving the first part of the claim. We proceed to show part 2.

**Part 2:** By Proposition 1, for every $x \in X$, there exists a $\vec{S}$-respecting arithmetic circuit $C_x$ such that for every $(\{\alpha_{i,b} \cdot \tilde{B}_{i,b}\}_{i\in[m],b\in\{0,1\}}, \mathbf{t}) \leftarrow \mathsf{Rand}(BP,p)$,

$$C_x(\{\alpha_{i,b} \cdot \tilde{B}_{i,b}\}_{i\in[m],b\in\{0,1\}}, \mathbf{t}) = \alpha_x \cdot p_x(\{\tilde{B}_{i,x[\mathsf{inp}(i)]}\}_{i\in[m]}, \mathbf{t})$$

where $\alpha_x = (\prod_{i\in[m]} \alpha_{i,x[\mathsf{inp}(i)]})$ and $C = \sum_{x\in X} C_x$. In particular, $p_x(\{\tilde{B}_{i,x[\mathsf{inp}(i)]}\}_{i\in[m]}, \mathbf{t}) = 0$ iff $C_x(\{\alpha_{i,b} \cdot \tilde{B}_{i,b}\}_{i\in[m],b\in\{0,1\}}, \mathbf{t}) = 0$ (since $\alpha_{i,b}$ is non-zero).

Thus, we have that

$$Pr[C(\mathsf{Rand}(BP,p))) = 0] = Pr[C_x(\mathsf{Rand}^{\alpha}(\mathsf{Rand}^{B}(BP,p))) = 0] = Pr[p_x(\mathsf{proj}_x(\mathsf{Rand}^{B}(BP,p))) = 0]$$

There must exist an input $x \in X$ such that $Pr[C_x(\mathsf{Rand}(BP,p))) = 0] < 1$ or else $Pr[C(\mathsf{Rand}(BP,p))) = 0] = 1$. By the first part of the claim, it follows that

$$Pr[C(\mathsf{Rand}(BP,p))) = 0] \leq 16wm/p,$$

which concludes the proof. □

Now we analyze the correctness of the simulator $\mathsf{CSim}$. We consider the following two cases: when $C(\mathsf{Rand}(BP,p))$ is always zero, and otherwise.

**Case 1:** $Pr[C(\mathsf{Rand}(BP,p)) = 0] = 1$**:** In this case we will show that the simulation always succeeds. If $Pr[C(\mathsf{Rand}(BP,p)) = 0] = 1$ then by Claim 16, for every $x \in X$, $Pr[p_x(\mathsf{proj}_x(\mathsf{Rand}^{B}(BP,p))) = 0] = 1$. Recall that $\mathsf{KSim}(1^m, p, v_{BP(x)})$ simulates $\mathsf{proj}_x(\mathsf{Rand}^{B}(BP,p))$ perfectly. Therefore, $\mathsf{CSim}$ always outputs 1 and hence succeeds.

**Case 2:** $Pr[C(\mathsf{Rand}(BP,p)) = 0] < 1$**:** In this case, by the first part of Claim 17 we have that

$$Pr[\mathsf{isZero}(C(\mathsf{Rand}(BP,p))) = 1] \leq 16wm/p$$

By the perfect simulation of $\mathsf{KSim}$, we have that

$$Pr[\mathsf{CSim}^{BP} = 1] = Pr[\forall x \ (d_x \leftarrow \mathsf{proj}_x(\mathsf{Rand}^{B}(BP,p)) : p_x(d_x) = 0)]$$

By second part of Claim 17 there exists input $x_C$ such that $Pr[p_{x_C}(\mathsf{proj}_{x_C}(\mathsf{Rand}^{B}(BP,p))) = 0] \leq 16wm/p$. Therefore,

$$Pr[\mathsf{CSim}^{BP} = 1] \leq Pr[p_{x_C}(\mathsf{proj}_{x_C}(\mathsf{Rand}^{B}(BP,p))) = 0] \leq 16wm/p$$

Therefore, by a union bound we have that

$$Pr[\mathsf{isZero}(C(\mathcal{D})) = \mathsf{CSim}^{BP} = 0] > 1 - 32wm/p$$

This concludes the proof of the lemma. □

□

**Remark 1.** In the above proof, we can rely on a weaker *entropic* notion of semantic security, where security holds only for message samplers that sample $\vec{m}_0$, $\vec{m}_1$ and $\vec{z}$ with high entropy. In particular, for our proof to go through it suffices to restrict to message samplers that ensure that the the entropy of $\vec{m}_0$ and $\vec{m}_1$, conditioned on $\vec{z}$ is "very high". This follows from the observation that the message sampler considered in the above proof has this property: Recall that $M$ outputs $\vec{m}_0 = \{(\alpha_{i,0}\cdot\tilde{B}_{i,0}, \alpha_{i,1}\cdot\tilde{B}_{i,1})\}_{i\in l_1,l_2}$ for two levels $l_1, l_2$, where $(\{\alpha_{i,b}\cdot\tilde{B}_{i,b}\}_{i\in[m'],b\in\{0,1\}}, \mathbf{t}) \leftarrow \mathsf{Rand}(BP, p)$ for some length $m'$ branching program $BP$. $\vec{z}$ contains $(\alpha_{i,0}\cdot\tilde{B}_{i,0}, \alpha_{i,1}\cdot\tilde{B}_{i,1})$, for all other levels $i \notin \{l_1, l_2\}$, and $\mathbf{t}$. Also recall that for every $i \in [m']$ and $b \in \{0,1\}$, $\tilde{B}_{i,b} = R_{i-1}B_{i,b}R_i^{-1}$ where $\{R_i\}_{i\in[m']}$ are random invertible matrices (and $R_0 = I_{5\times 5}$) and $\{B_{i,b}\}_{i\in[m'],b\in\{0,1\}}$ are the matrices of $BP$. It easily follows (using the same argument as [Kil88]) that even conditioned on $\{\alpha_{i,b}, \tilde{B}_{i,b}\}_{i\notin\{l_1,l_2\},b\in\{0,1\}}$ and the whole of $R_{m'}$ (as opposed to just $\mathbf{t}$), $\tilde{B}_{l_1,0}$ is a random invertible matrix, and thus $\vec{m}_0$ has at least the entropy of a random invertible matrix, conditioned on $\vec{z}$. The same argument applies to $\vec{m}_1$.

### 5.4 Achieving Obfuscation for Arbitrary Programs

[GGH$^+$13b] show that any indistinguishability obfuscation scheme for $\mathsf{NC}^1$ can be bootstrapped into an indistinguishability obfuscation scheme for all poly-sized circuits using FHE. That is, they prove the following theorem.

**Theorem 17** ([GGH$^+$13b]—informally stated)**.** *Assume the existence of indistinguishability obfuscators $i\mathcal{O}$ for $\mathsf{NC}^1$ and a leveled Fully Homomorphic Encryption scheme with decryption in $\mathsf{NC}^1$. Then there exists an indistinguishability obfuscator $i\mathcal{O}'$ for arbitrary poly-sized circuits.*

Applying their construction to our indisinguishability obfuscator yields an indistinguishability obfuscator for arbitrary polynomial size circuits:

**Theorem 18.** *Assume the existence of semantically secure multi-linear encodings and a leveled Fully Homomorphic Encryption scheme with decryption in $\mathsf{NC}^1$. Then there exists an indistinguishability obfuscators for arbitrary poly-sized circuits.*

## 6 Acknowledgments

We are very grateful to Kai-min Chung for many helpful conversations.

## References

[ABG$^+$13] Prabhanjan Ananth, Dan Boneh, Sanjam Garg, Amit Sahai, and Mark Zhandry. Differing-inputs obfuscation and applications. 2013.

[Bar86] David A. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in nc$^1$. In *STOC*, pages 1–5, 1986.

[BCP13] Elette Boyle, Kai-Min Chung, and Rafael Pass. On extractability obfuscation. Technical report, Cryptology ePrint Archive, Report 2013/650, 2013. http://eprint. iacr. org, 2013.

[BCPR13] Nir Bitansky, Ran Canetti, Omer Paneth, and Alon Rosen. Indistinguishability obfuscation vs. auxiliary-input extractable functions: One must fall. Technical report, Cryptology ePrint Archive, Report 2013/641, 2013. 5, 2013.

[BGI$^+$01] Boaz Barak, Oded Goldreich, Rusell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im) possibility of obfuscating programs. In *Advances in CryptologyâĂŤCRYPTO 2001*, pages 1–18. Springer, 2001.

[BGK+13] Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Protecting obfuscation against algebraic attacks. Cryptology ePrint Archive, Report 2013/631, 2013.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *ITCS*, pages 309–325, 2012.

[BP13] Elette Boyle and Rafael Pass. Limits of extractability assumptions with distributional auxiliary input. 2013.

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

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. In *FOCS*, pages 97–106, 2011.

[BZ13] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. Technical report, Cryptology ePrint Archive, Report 2013/642, 2013. http://eprint. iacr. org, 2013.

[Gen09] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009.

[GGH13a] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In *Advances in Cryptology–EUROCRYPT 2013*, pages 1–17. Springer, 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. *Proc. of FOCS 2013*, 2013.

[GGHR13] Sanjam Garg, Craig Gentry, Shai Halevi, and Mariana Raykova. Two-round secure mpc from indistinguishability obfuscation. Technical report, IACR Cryptology ePrint Archive, 2013: 601, 2013.

[Had00] Satoshi Hada. Zero-knowledge and code obfuscation. In *Advances in Cryptology–ASIACRYPT 2000*, pages 443–457. Springer, 2000.

[HSW13] Susan Hohenberger, Amit Sahai, and Brent Waters. Replacing a random oracle: Full domain hash from indistinguishability obfuscation. Technical report, Cryptology ePrint Archive, Report 2013/509, 2013. http://eprint. iacr. org, 2013.

[Kil88] Joe Kilian. Founding crytpography on oblivious transfer. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 20–31. ACM, 1988.

[LCT+13] Tancrède Lepoint, Jean-Sébastien Coron, Mehdi Tibouchi, et al. Practical multilinear maps over the integers. In *CRYPTO 2013-33rd Annual Cryptology Conference Advances in Cryptology*, volume 8042, pages 476–493, 2013.

[Nao03] Moni Naor. On cryptographic assumptions and challenges. In *CRYPTO*, pages 96–109, 2003.

[RAD78] R L Rivest, L Adleman, and M L Dertouzos. On data banks and privacy homomorphisms. *Foundations of Secure Computation, Academia Press*, pages 169–179, 1978.

[SW13] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: Deniable encryption, and more. Technical report, IACR Cryptology ePrint Archive, 2013: 454, 2013.

## A Technical Lemma

**Claim 19.** *Fix $m, w \in \mathbb{N}$, and let $p \in \mathbb{N}$ be a prime. Let $\mathcal{D}_0$ be the following distribution:*

$$\mathcal{D}_0 = \{\{R_i\}_{i\in[m]}, \{\alpha_{i,b}\}_{i\in[m],b\in\{0,1\}}\}$$

*where each $R_i$ is a uniformly random invertible matrix in $\mathbb{Z}_p^{w\times w}$ (i.e $det(R_i) \neq 0$, and each $\alpha_{i,b}$ is a uniformly random non-zero scalar in $\mathbb{Z}_p$.*

*Let $\mathcal{D}_1$ be a distribution defined identically to $\mathcal{D}_0$, except with each $R_i$ being a uniformly random (not necessarily invertible) matrix in $\mathbb{Z}_p^{w\times w}$, and each $\alpha_{i,b}$ a uniformly random (not necessarily non-zero) scalar in $\mathbb{Z}_p$.*
*Then:*

$$\Delta(\mathcal{D}_0, \mathcal{D}_1) \leq 8wm/p$$

*where $\Delta(\mathcal{D}_0, \mathcal{D}_1)$ denotes the statistical distance between distributions $\mathcal{D}_0$ and $\mathcal{D}_1$.*

*Proof.* Note that $\mathcal{D}_0$ and $\mathcal{D}_1$ are each uniformly distributed on their respective supports, and that $\mathsf{supp}(\mathcal{D}_0) \subseteq \mathsf{supp}(\mathcal{D}_1)$. Then the statistical distance between $\mathcal{D}_0$ and $\mathcal{D}_1$ can be computed as follows:

$$\begin{aligned}
\Delta(\mathcal{D}_0, \mathcal{D}_1) &= \sum_{d\in\mathsf{supp}(\mathcal{D}_0)\cup\mathsf{supp}(D_1)} |\Pr[\mathcal{D}_0 = d] - \Pr[\mathcal{D}_1 = d]| \\
&= \sum_{d\in\mathsf{supp}(\mathcal{D}_0)} |\Pr[\mathcal{D}_0 = d] - \Pr[\mathcal{D}_1 = d]| + \sum_{d\in\mathsf{supp}(\mathcal{D}_1)\setminus\mathsf{supp}(\mathcal{D}_0)} |\Pr[\mathcal{D}_1 = d]| \\
&= \sum_{d\in\mathsf{supp}(\mathcal{D}_0)} |\frac{1}{|\mathsf{supp}(\mathcal{D}_0)|} - \frac{1}{|\mathsf{supp}(\mathcal{D}_1)|}| + \sum_{d\in\mathsf{supp}(\mathcal{D}_1)\setminus\mathsf{supp}(\mathcal{D}_0)} |\frac{1}{|\mathsf{supp}(\mathcal{D}_1)|}| \\
&= (|\mathsf{supp}(\mathcal{D}_0)| \cdot |\frac{1}{|\mathsf{supp}(\mathcal{D}_0)|} - \frac{1}{|\mathsf{supp}(\mathcal{D}_1)|}|) + (|\mathsf{supp}(\mathcal{D}_1) \setminus \mathsf{supp}(\mathcal{D}_0)| \cdot |\frac{1}{|\mathsf{supp}(\mathcal{D}_1)|}|) \\
&= 2 \cdot (1 - \frac{|\mathsf{supp}(\mathcal{D}_0)|}{|\mathsf{supp}(\mathcal{D}_1)|})
\end{aligned}$$

But notice that $(1 - \frac{|\mathsf{supp}(\mathcal{D}_0)|}{|\mathsf{supp}(\mathcal{D}_1)|})$ can be interpreted as $\Pr[\exists i \in [m], b \in \{0,1\} : \det(R_i) = 0 \vee \alpha_{i,b} = 0]$. For each $i \in [m]$, the probability $det(R_i) = 0$ can be bounded by applying the Schwartz-Zippel lemma to the $det(\cdot)$, which is a polynomial of degree $w$. Thus we have that $\Pr[det(R_i) = 0] \leq w/p$. Further, each $\alpha_{i,b}$ is zero with probability $1/p$. Hence, applying a union bound, we have that

$$\begin{aligned}
\Delta(\mathcal{D}_0, \mathcal{D}_1) &= 2 \cdot (1 - \frac{|\mathsf{supp}(\mathcal{D}_0)|}{|\mathsf{supp}(\mathcal{D}_1)|}) \\
&\leq 2 \cdot (2m/p + mw/p) \\
&\leq 8wm/p
\end{aligned}$$

□

## B Proof of Proposition 1

**(i)** Our proof closely follows the corresponding decomposition algorithms in [BGK+13], with the following differences:

- [BGK+13]'s decomposition additionally shows that $X$ is polynomial-sized, relying on some extra features in their construction, namely dual-input branching programs, and a modified collection of sets chosen by SetSystem.

- We use the definition of set-respecting circuits to slightly simplify the proof.

In order to prove the decomposition of $C$ into the sum of $C_x$, we will define a recursive algorithm $\mathsf{Decomp}$ that can compute this decomposition. Before we do so, however, we define some notation to help us denote that a circuit respects a single input $x \in \{0,1\}^n$. For any circuit $C$ that has its input wires labelled with sets in $\vec{S}$, let the *profile* of $C$, denoted $\mathsf{prof}(C)$, be a string $x \in \{0,1,*\}^n \cup \{\bot\}$, defined as follows:

- $x = \bot$ if for some $j \in [n]$, $C$ contains both an input wire labelled with with a set from $\{S_{i,0} : \mathsf{inp}(i) = j\}$, and an input wire labelled with a set from $\{S_{i,1} : \mathsf{inp}(i) = j\}$.
- otherwise:
  - $x[j] = b$ if some input wire of $C$ is labelled with with a set from $\{S_{i,b} : \mathsf{inp}(i) = j\}$, and no input wire is labelled with a set from $\{S_{i,(1-b)} : \mathsf{inp}(i) = j\}$.
  - $x[j] = *$ if no input wire of $C$ is labelled with a set from either of $\{S_{i,b} : \mathsf{inp}(i) = j\}$ or $\{S_{i,(1-b)} : \mathsf{inp}(i) = j\}$.

We say that $\mathsf{prof}(C)$ is *consistent* if $\mathsf{prof}(C) \neq \bot$. We say that $\mathsf{prof}(C)$ is *complete and consistent* if $\mathsf{prof}(C)$ is not $\bot$, and it does not contain any $*$ characters, that is, $\mathsf{prof}(C) \in \{0,1\}^n$ . Notice that if $\mathsf{prof}(C)$ is complete and consistent, then $C$'s input wires are labelled only with sets respecting a single input $x = \mathsf{prof}(C) \in \{0,1\}^n$, that is, only with sets $\in \{S_{i,x[\mathsf{inp}(i)]}\}_{i\in[m]} \cup \{S_t\}$.

We further define $S(C)$ to be the label on the output wire of $C$.

The algorithm $\mathsf{Decomp}$ will take as input an arithmetic circuit $C$, and return a set $X \subseteq \{0,1,*\}^n \cup \{\bot\}$, together with a set of circuits $L = \{C_x)\}_{x\in X}$ satisfying

- $\forall x \in X$, $\mathsf{prof}(C_x) = x$
- $C = \sum_{x\in X} C_x$

We will later show that if $C$ is $\vec{S}$-respecting, then each $C_x$ is also $\vec{S}$-respecting, and further, that each $C_x$ has a complete and consistent profile. This implies that, when $C$ is $\vec{S}$-respecting, that $X \subseteq \{0,1\}^n$.

We define $\mathsf{Decomp}(C)$ recursively, as follows:

- When $C$ is a single input wire, if $C$ is labelled with $S_{i,b}$, $\mathsf{Decomp}$ outputs $X = \{\mathsf{prof}(C)\}$ and the singleton set $\{C\}$.
- When $C$ is of the form $C_1 + C_2$, then $\mathsf{Decomp}$ computes $X_1, L_1 = \mathsf{Decomp}(C_1)$ and $X_2, L_2 = \mathsf{Decomp}(C_2)$, and sets $X = X_1 \cup X_2$ and $L = L_1 \cup L_2$. If $L$ contains $C_x$ and $C_{x'}$ with the same profile, that is, $x = x'$, then $\mathsf{Decomp}$ replaces the two circuits with the single circuit $(C_x + C_{x'})$, repeating this process until all the circuits in $L$ have distinct profiles, and outputs $X$ and $L$. The case for $C = C_1 - C_2$ follows analogously, except subtracting two circuits with the same profile rather than adding them.
- When $C$ is of the form $C_1 \cdot C_2$, then $\mathsf{Decomp}$ recursively obtains sets $X_1, L_1 = \mathsf{Decomp}(C_1)$ and $X_2, L_2 = \mathsf{Decomp}(C_2)$. For each element $C_x \in L_1$ and $C_{x'} \in L_2$, $\mathsf{Decomp}$ determines $C_{x''} = C_x \cdot C_{x'}$, where $x'' = \mathsf{prof}(C_x \cdot C_{x'})$ and adds $x''$ to $X$ and $C_{x''}$ to $L$. $\mathsf{Decomp}$ combines circuits with the same profile in $L$ as described above, so that each circuit in $L$ has a distinct profile, and outputs $X$ and $L$.

By examining the description of $\mathsf{Decomp}$, we see that the following properties hold inductively, at each recursive level of $\mathsf{Decomp}$:

- $C = \sum_{x \in X} C_x$
- The output wire of each $C_x$ has the same label as the output wire of $C$, that is, $S(C_x) = S(C)$
- If $C$ only makes additions and multiplications respecting $\vec{S}$, then each $C_x$ also only makes additions and multiplications respecting $\vec{S}$.

From the last two properties, we infer that if $C$ is $\vec{S}$-respecting, then each $C_x$ is also $\vec{S}$-respecting. Further, if $C$ is $\vec{S}$-respecting, then no $x \in X$ contains '$*$', since in order to be $\vec{S}$-respecting, each $C_x$ must have its output wire labelled with $[k]$, and thus must have at least one input wire labelled with some entry of $\mathbb{S}^j$, for each $j \in [n]$.

We finally argue that if $C$ is $\vec{S}$-respecting, then $\perp \notin X$. Assume for contradiction that $\perp \in X$, that is, there is some $\vec{S}$-respecting $C_\perp$ output by $\mathsf{Decomp}(C)$. We will show that such a $C_\perp$ cannot exist, using the properties of the straddling set system.

Let $C'_\perp$ be the first sub-circuit of $C_\perp$ that has profile $\perp$. That is, all subcircuits of $C'_\perp$ have profiles $\neq \perp$, but $C'_\perp$ has profile $\perp$. Since $C_\perp$ is output by $\mathsf{Decomp}$, at some recursive step of $\mathsf{Decomp}$, $C'_\perp$ must have been added to the set of circuits $L$ output by $\mathsf{Decomp}$ at that step.

Then this step must be a multiplication step, since addition (or subtraction) steps never introduce new profiles because they only add (or subtract) circuits with the same profiles, and in the base case, single input wires never have $\perp$ as their profile. Thus, at this step, $\mathsf{Decomp}$ must multiply together two circuits $C_x$ and $C_{x'}$ such that, for some $j$, $C_x$ has an input wire labelled with $S_{i,b}$, and $C_{x'}$ has an input wire labelled with $S_{i',1-b}$ for $i, i'$ with $\mathsf{inp}(i) = \mathsf{inp}(i') = j$.

We now use the following claim, along the lines of Claim 4 in [BGK+13]:

**Claim 20.** *Let $C$ be $\vec{S}$-respecting. If $C'$ is a sub-circuit of $C$ and $\mathcal{T}' \subseteq \mathbb{S}^j$ is an exact cover of $S(C') \cap U_j$, then there exists an exact cover $\mathcal{T}$ of $S(C) \cap U_j$ such that $\mathcal{T}' \subseteq \mathcal{T}$.*

*Proof.* The proof is by induction. If $C$ is of the form $C_1 + C_2$, and $\mathcal{T}_1 \subseteq \mathbb{S}^j$ is an exact cover of $S(C_1) \cap U_j$, then $\mathcal{T}_1$ is also an exact cover of $S(C) \cap U_j$, since $S(C) = S(C_1)$, and similarly for $C_2$.

If $C$ is of the form $C_1 \cdot C_2$, and $\mathcal{T}_1, \mathcal{T}_2 \subseteq \mathbb{S}^j$ are exact covers of $S(C_1) \cap U_j$ and $S(C_2) \cap U_j$ respectively, then since $S(C_1) \cap S(C_2) = \emptyset$ and since $S(C) = S(C_1) \cup S(C_2)$ then $\mathcal{T}_1 \cup \mathcal{T}_2$ is an exact cover of $S(C) \cap U_j$. ☐

Applying the above claim, we see that since $C_x$ has an input wire labelled with $S_{i,b}$ as a "sub-circuit", then there exists an exact cover of $S(C_x) \cap U_j$ that contains $S_{i,b}$. Similarly, there exists an exact cover of $S(C_{x'}) \cap U_j$ that contains $S_{i',1-b}$. Since $C'_\perp = C_x \cdot C_{x'}$ and this multiplication is $\vec{S}$-respecting, there exists an exact cover of $S(C'_\perp) \cap U_j$ that contains both $S_{i,b}$ and $S_{i',1-b}$. But then there exists an exact cover of $S(C_\perp) \cap U_j$ that also contains both $S_{i,b}$ and $S_{i',1-b}$. But since $C_\perp$ is $\vec{S}$-respecting, $S(C_\perp) \cap U_j = U_j$. However, since $\mathbb{S}^j$ is a straddling set system, we know that no exact cover of $U_j$ can contain both $S_{i,b}$ and $S_{i',1-b}$. This contradicts the existence of $C_\perp$, and so $\perp \notin X$.

We thus have that each $x \in X$ must be such that $x \in \{0,1\}^n$, implying that each $C_x$ has a complete and consistent profile. Hence we have that that $X \subseteq \{0,1\}^n$, that $C \equiv \sum_{x \in X} C_x$, that each $C_x$ is $\vec{S}$-respecting, and, from the earlier discussion, that each $C_x$ has input wires using only sets $\in \{S_{i,x[\mathsf{inp}(i)]}\}_{i \in [m]} \cup \{S_t\}$.

**(ii)** Consider each circuit $C_x$ in the decomposition described above. We will show that each $C_x$ can be decomposed into $\alpha_x$ and $p_x$ as defined in the proposition statement. Since $C_x$ is an arithmetic circuit, we can write $C_x$ as a polynomial, and in fact, as the sum of monomials $s_x$ (possibly exponentially many), so that $C_x = \sum s_x$. Since $C_x$ is $\vec{S}$-respecting, each $s_x$ is $\vec{S}$-respecting also. Further, since $s_x$ is a monomial, it can be represented as an arithmetic circuit consisting only of multiplication gates. Further, it must have exactly $m + 1$ input wires, 1 corresponding to each level of the branching program, and 1 corresponding to $t$, since this is the only way to obtain an output wire with label $[k]$ using only multiplication gates.

But since $C_x$ only has input labels from sets $\in \{S_{i,x[\mathsf{inp}(i)]}\}_{i\in[m]} \cup \{S_t\}$, so must each $s_x$, and thus the inputs to $s_x$ must consist of a choice of a single element from each of $\{\alpha_{i,x[\mathsf{inp}(i)]} \cdot \tilde{B}_{i,x[\mathsf{inp}(i)]}\}_{i\in[m]} \cup \{\mathbf{t}\}$. But then each $s_x$ can be written as $\alpha_x \cdot \tilde{s}_x(\{\tilde{B}_{i,x[\mathsf{inp}(i)]}\}_{i\in[m]}, \mathbf{t})$, where $\tilde{s}_x$ contains exactly one term from each $\tilde{B}_{i,x[\mathsf{inp}(i)]}$, and one term from $\mathbf{t}$. Hence $C_x = \sum s_x$ can also be written in the form $C_x = \alpha_x \cdot p_x(\{\tilde{B}_{i,x[\mathsf{inp}(i)]}\}_{i\in[m]}, \mathbf{t})$, where $p_x = \sum \tilde{s}_x$. Further, when $p_x$ is viewed as a sum of monomials, each monomial contains exactly one term from each $\tilde{B}_{i,x[\mathsf{inp}(i)]}$, and from $\mathbf{t}$.