# Optimizing Obfuscation: Avoiding Barrington's Theorem

Prabhanjan Ananth Divya Gupta Yuval Ishai Amit Sahai*

**Abstract**

In this work, we seek to optimize the efficiency of secure general-purpose obfuscation schemes. We focus on the problem of optimizing the obfuscation of general Boolean formulas – this corresponds to optimizing the "core obfuscator" from the work of Garg, Gentry, Halevi, Raykova, Sahai, and Waters (FOCS 2013), and all subsequent works constructing general-purpose obfuscators. This core obfuscator builds upon approximate multilinear maps, where efficiency in proposed instantiations is closely tied to the maximum number of "levels" of multilinearity required. The most efficient previous construction of a core obfuscator, due to Barak, Garg, Kalai, Paneth, and Sahai (Eurocrypt 2014) required the maximum number of levels of multilinearity to be $\Theta(\ell s^{6.82})$, where $s$ is the size of the Boolean formula to be obfuscated, and $\ell$ is the number of input bits to the formula. In contrast, our construction only requires the maximum number of levels of multilinearity to be $\Theta(\ell s)$. This results in significant improvements in both the total size of the obfuscation, as well as the running time of evaluating an obfuscated formula.

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

# 1 Introduction

The goal of general-purpose program obfuscation is to make an arbitrary computer program "unintelligible" while preserving its functionality. At least as far back as the work of Diffie and Hellman in 1976 [DH76][1], researchers have contemplated applications of general-purpose obfuscation. As part of their systematic study of obfuscation, Barak, Goldreich, Impagliazzo, Rudich, Sahai, Vadhan, and Yang in 2001 [BGI+01][2] also enumerated several additional applications of general-purpose obfuscation, ranging from software intellectual property protection and removing random oracles, to eliminating software watermarks. However, until 2013, even heuristic constructions for general-purpose obfuscation were not known.

This changed with the work of Garg, Gentry, Halevi, Raykova, Sahai, and Waters in 2013 [GGH+13b], which gave the first candidate construction for a general-purpose obfuscator. This work also showed how to use general-purpose obfuscation to obtain the first general-purpose functional encryption scheme. Soon thereafter, Sahai and Waters [SW14] showed how to use general-purpose obfuscation to build many interesting cryptographic primitives, including the first fully deniable encryption scheme, resolving a 16-year-old open problem. Since then, the floodgates have opened, and many new applications of general-purpose obfuscation have been explored [BCPR13, BP13, MR13, BCP13, ABG+13, MO13, GJKS13, PPS13, GGHW13, GGJ+14, BBC+14, BFM14, KNY14, GHRW14, HSW14, BR14a, BR14b, GGHR14, CGK14].

**Efficiency of General-Purpose Obfuscation.** This great interest in the utility of obfuscation leads to a natural and pressing goal: to improve the efficiency of general-purpose obfuscation. Up to this point, the simplest and most efficient proposed general-purpose obfuscator was given by [BGK+14], building upon [GGH+13b, BR14b]. However, the general-purpose obfuscator presented in [BGK+14] (see below for more details) remains extremely inefficient.

Our work aims to initiate a systematic research program into improving the efficiency of general-purpose obfuscation. Tackling this important problem will no doubt be the subject of many research papers to come. We begin by recalling the two-stage approach to general-purpose obfuscation outlined in [GGH+13b] and present in all subsequent work on constructing general-purpose obfuscators:

1. At the heart of their construction is the "core obfuscator" for Boolean formulas (i.e., $\mathbf{NC}^1$ circuits), building upon a simplified subset of the Approximate Multilinear Maps framework of Garg, Gentry, and Halevi [GGH13a] that they call Multilinear Jigsaw Puzzles. (We will defer discussion of security to later.)
2. Next, a way to bootstrap from the core obfuscator for Boolean formulas to general circuits is used. The work of [GGH+13b, BR14b, BGK+14] all adopt a method for bootstrapping using Fully Homomorphic Encryption. This bootstrapping method works provably with the security definition of indistinguishability obfuscation, under the well-studied LWE assumption. Alternatively, the earlier work of [GIS+10] constructed a universal stateless hardware token for obfuscation that can be implemented by a Boolean formula. Applebaum [App13] gives a simpler alternative construction that has the disadvantage of requiring the size of the Boolean formula to be polynomial in the input size and security parameter (rather than only in the security parameter in [GIS+10]). Using either of these alternative approaches [GIS+10, App13], however, requires an ad-hoc (but arguably natural) assumption to bootstrap from obfuscation for Boolean formulas to indistinguishability

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

[2] The work of [BGI+01] is best known for their constructions of "unobfuscatable" classes of functions $\{f_s\}$ that roughly have the property that given *any* circuit evaluating $f_s$, one can extract the secret $s$, yet given only *black-box* access to $f_s$, the secret $s$ is hidden.

obfuscation for general circuits.

Our work focuses on improving the efficiency of first of these steps: namely, the core obfuscator for Boolean formulas. All previous constructions of a core obfuscator [GGH$^+$13b, BR14b, BGK$^+$14] first apply Barrington's theorem to convert the Boolean formula into an equivalent "matrix branching program," which is then obfuscated. For any formula of depth $d$, Barrington's theorem gives a constant width matrix branching program of length $4^d$. Since Barrington's theorem outputs a branching program whose length is exponential in the depth of the formula, to avoid blow up in size, it is crucial to balance the depth of the formula. Hence, the first step would be to balance the formula to get a depth which is logarithmic in the size and then apply Barrington's theorem. For general formulas of size $s$, the best known depth obtained by balancing them is at least $3.14 \log s$ [Spi71]. There have been other works which try to optimize the *size* of balanced formulas [BB94], but the depth of the formula obtained by these works is worse. Thus, the matrix branching program obtained by applying Barrington's theorem [Bar86] to a formula of depth $3.14 \log s$ has length $s^{6.82}$. This is a major cause of inefficiency. In particular, the bound of $s^{6.82}$ on the length of the branching program not only affects the number of elements given out as the final obfuscation, but also number of levels of multi-linearity required by the scheme. Since the size of the each multilinear encoding grows with the number of levels of multilinearity required in known realizations of approximate multilinear maps [GGH13a, CLT13], this greatly affects the size of the final obfuscated program and also the evaluation time. Hence, in order to optimize the size of obfuscation it is critical to find an alternative approach.

**Our Contributions.** In our work, we posit an alternative strategy for obfuscation that avoids Barrington's theorem, and the need to balance Boolean formulas at all. A crucial first step is to formulate a notion of a "relaxed matrix branching program[3]," which relaxes some of the requirements of matrix branching programs needed in [GGH$^+$13b, BR14b, BGK$^+$14]. Nevertheless, we show how to adapt the construction and security proofs of [BGK$^+$14] to work with our relaxed matrix branching programs.

Armed with our relaxed notion of matrix branching programs, we then seek to find ways to convert Boolean formulas into such relaxed matrix branching programs without needing to invoke Barrington's theorem. We do so in a modular way. We provide a general and efficient transformation that converts any ordinary graph-based layered branching program, satisfying certain technical conditions, into a relaxed branching program. We then turn to the (abundant) literature on ordinary branching programs to find efficient ways to represent Boolean formulas as graph-based layered branching programs, and verify that these representations satisfy our technical conditions.

We begin by considering formulas consisting of only AND and NOT gates. We give a simple construction of the folklore theorem that converts any such formula of size $s$ into a layered graph-based branching program. This yields a relaxed matrix branching program consisting of $O(s)$ square matrices of dimension $O(s)$. This gives an overall size of the relaxed matrix branching program to be $O(s^3)$, in contrast to $O(s^{6.82})$ obtained using balancing and Barrington's theorem. Note that one more step is needed by us, as well as previous work [GGH$^+$13b, BR14b, BGK$^+$14], to make the matrix branching programs *input-oblivious*. This incurs an additional multiplicative overhead of $\ell$.

Another important dimension of comparison, as mentioned before, is levels of multi-linearity required, which in turn equals the length of the matrix branching program. In our construction, because the length of our relaxed matrix branching program is only $O(\ell \cdot s)$, we require the multi-linearity to be only $O(\ell \cdot s)$ in contrast with $O(\ell \cdot s^{6.82})$ in previous work [GGH$^+$13b, BR14b, BGK$^+$14].

[3]We discuss this notion of relaxed matrix branching program formally in Section 2.3.

**Table 1** Comparing the Efficiency of Obfuscation Schemes for formulas over AND and NOT gates. $\lambda$ is the security parameter of the scheme, $s$ is the formula size, and $\ell$ is the size of the input.

| Scheme | [GGH$^+$13b],[BGK$^+$14] | This Work |
|---|---|---|
| Size of matrix branching program | $O(\ell s^{6.82})$ | $O(\ell s^3)$ |
| Levels of Multi-linearity | $O(\ell s^{6.82})$ | $O(\ell s)$ |
| Size of Obfuscation (Aggressive Setting) | $O(\ell^2 s^{13.64}\lambda^{c'})$ | $O(\ell^2 s^4 \lambda^{c'})$ |
| Size of Obfuscation (Conservative Setting) | $O(\ell^3 s^{20.46}\lambda^{c})$ | $O(\ell^3 s^5 \lambda^{c})$ |
| Evaluation Time (Number of Multilinear Operations) | $O(\ell s^{6.82})$ | $O(\ell s^{3.37})$ |
| Evaluation Time (Aggressive Setting) | $\tilde{O}(\ell^2 s^{13.64}\lambda^{c'})$ | $\tilde{O}(\ell^2 s^{4.37}\lambda^{c'})$ |
| Evaluation Time (Conservative Setting) | $\tilde{O}(\ell^3 s^{20.46}\lambda^{c})$ | $\tilde{O}(\ell^3 s^{5.37}\lambda^{c})$ |

**Table 2** Comparing the Efficiency of Obfuscation Schemes for formulas over AND, XOR, and NOT gates. $\lambda$ is the security parameter of the scheme, $s$ is the formula size, and $\ell$ is the size of the input.

| Scheme | [GGH$^+$13b],[BGK$^+$14] | This Work |
|---|---|---|
| Size of matrix branching program | $O(\ell s^{6.82})$ | $O(\ell s^{3.39})$ |
| Levels of Multi-linearity | $O(\ell s^{6.82})$ | $O(\ell s)$ |
| Size of Obfuscation (Aggressive Setting) | $O(\ell^2 s^{13.64}\lambda^{c'})$ | $O(\ell^2 s^{4.39}\lambda^{c'})$ |
| Size of Obfuscation (Conservative Setting) | $O(\ell^3 s^{20.46}\lambda^{c})$ | $O(\ell^3 s^{5.39}\lambda^{c})$ |
| Evaluation Time (Number of Multilinear Operations) | $O(\ell s^{6.82})$ | $O(\ell s^{3.84})$ |
| Evaluation Time (Aggressive Setting) | $\tilde{O}(\ell^2 s^{13.64}\lambda^{c'})$ | $\tilde{O}(\ell^2 s^{4.84}\lambda^{c'})$ |
| Evaluation Time (Conservative Setting) | $\tilde{O}(\ell^3 s^{20.46}\lambda^{c})$ | $\tilde{O}(\ell^3 s^{5.84}\lambda^{c})$ |

The recommended parameter settings in all published works on multilinear encodings [GGH13a, CLT13] show that for $\kappa$-level multilinear encodings, the size of each encoding is $O(\kappa^2 \cdot \lambda^c)$ for some constant $c$. We refer to this as the "conservative" setting of parameters. Using conservative setting of parameters, we get size of each encoding to be $O(\ell^2 s^2 \lambda^c)$ compared to $O(\ell^2 s^{13.64}\lambda^c)$ in previous work [GGH$^+$13b, BR14b, BGK$^+$14]. Our reading of these papers suggests that there may be a setting of parameters where size of each encoding is $O(\kappa \cdot \lambda^{c'})$ which avoids all the known attacks for the case when no level-0 encodings are being given out. We refer to this setting of parameters as "aggressive" setting. Using aggressive setting of parameters, we get size of each encoding to be $O(\ell s \lambda^{c'})$ compared to $O(\ell s^{6.82}\lambda^{c'})$ in previous work [GGH$^+$13b, BR14b, BGK$^+$14]. Using this we get the size of the obfuscated formula to be $O(\ell^2 s^4 \lambda^{c'})$ in contrast with $O(\ell^2 s^{13.64}\lambda^{c'})$ using existing techniques. The improvement is even higher for the conservative setting as shown in Table 1. Moreover, the time of evaluation of the obfuscated program is crucially related to the size of the multilinear encodings. A summary of the comparison is provided in Table 1.

Next, we consider Boolean formulas over AND, NOT, and XOR gates. For this setting, we re-examine the work of [SWW99], which gives a transformation of formulas over AND, NOT, and XOR gates to a layered branching program. The graph-based branching program described in [SWW99] satisfies our conditions and can be used to obfuscate formulas over AND, XOR, and NOT gates. For a formula of size $s$, [SWW99] gives a layered branching program of size $O(s^{1.195})$ such that the number of layers is $O(s)$. Size of the obfuscation obtained using these branching programs and other parameters have been summarized in Table 2. We note that the Barrington's theorem based approach also works for the formulas having AND, NOT, and XOR gates [Bon].

**Security.** While improving security of obfuscation is not the focus of this work, our work on improving efficiency of obfuscation would be meaningless if it sacrificed security. We give evidence for the security of our constructions in the same way that the work of [BGK+14] does: by showing that our constructions achieve a strong virtual black-box notion of security, against adversaries that are limited to algebraic attacks allowed in a generic multilinear model. (Note that this argument also shows that the obfuscator achieves the indistinguishability obfuscation security definition in a generic multilinear model, as a special case.) This is essentially the strongest evidence of security for any general-purpose obfuscation scheme that we have to date.

# 2 Preliminaries

We denote the security parameter by $\lambda$. We denote by $[a, b]$ the intervals, defined only for $b > a$, to consist of all integers from $a$ to $b$. More formally, $[a, b] = \{a, a + 1, \ldots, b\}$. The interval $[1, a]$ is denoted by $[a]$.

Let $e_i^{(n)}$ denote a $1 \times n$ matrix with the $i^{th}$ entry being 1 and the rest of the entries being 0. When the size of the matrix is clear we will denote $e_i^{(n)}$ by $e_i$.

We now give some background for formulae and branching programs.

## 2.1 Boolean formulae

We first recall the notion of boolean circuits and then we will define boolean formulae. A boolean circuit corresponding to a function $f : \{0, 1\}^l \to \{0, 1\}$ is a directed acyclic graph (DAG). The vertices in this graph are of two types – input nodes and gates. The gates are of three types - AND, XOR and NOT. The indegree of a vertex is 0 for input nodes, 1 for the NOT gate and 2 for the AND and the XOR gate. The outdegree of an output gate is 0 and it is at least 1 for all other vertices. The *fan-out* of a gate is defined to be the number of output wires corresponding to that gate and *fan-in* of a gate is defined to be the number of input wires to that gate. In this work, we consider a special type of circuits called formulae. A boolean formula is a boolean circuit where the fan-out of each gate is 1. In graph theoretic terms, a formula is a tree. We define the *size* of a formula to be the number of gates in the formula. Further, we define a *subtree* in a formula to be the subgraph in a formula consisting of all paths from the input wires to that gate.

Throughout this paper, unless mentioned, the input length of a formula is $\ell$. We define *index* of an input wire $w$, denoted by $index(w)$, in the formula to be $j$ if the $j^{th}$ bit of the formula is fed into the wire $w$.

## 2.2 Layered branching programs

A layered branching program is a weighted directed acyclic graph with three designated vertices, namely, a source vertex, an accept vertex and a reject vertex. A source vertex has no incoming edges while the accept and the reject vertices have no outgoing edges. Each edge in the graph is labeled either $x_i = 0$ or $x_i = 1$ for some $i \in [\ell]$. Further, the set of vertices is partitioned into a sequence of layers such that the first layer in the sequence consists of just the source vertex while the last layer in the sequence consists of the accept and the reject vertices.

Associated to a branching program is an *evaluation* function, denoted by $\mathsf{inp}$, that is used to evaluate the program on an input. The evaluation function maps each layer of vertices to an input bit. The evaluation of a branching program on an input, say $x$, proceeds as follows. Suppose the evaluation function maps a layer $\mathcal{L}_i$ to $x_j$, where $x_j$ is the $j^{th}$ input bit of $x$. If $x_j = 0$, we remove the outgoing edges of $\mathcal{L}_i$ which are labeled 1 and similarly if $x_j = 1$ then we remove the outgoing edges which are labeled 0. We perform this for every layer in the branching program. The resulting graph is denoted by $\mathsf{BP}|_x$. If there is a path from the source vertex

to the accept vertex in $\mathsf{BP}|_x$ and no path exists from the source vertex to the reject vertex in the resulting graph then the output of the branching program on $x$, denoted by $\mathsf{BP}(x)$, is 1. Symmetrically, if there is a path from the source to the reject vertex in $\mathsf{BP}|_x$ and no path exists from the source to the accept vertex in $\mathsf{BP}|_x$ then $\mathsf{BP}(x) = 0$.

Now that we have defined branching programs, we establish some notation.

- We denote the source vertex in a branching program $\mathsf{BP}$ to be $\mathsf{BP}[\mathsf{source}]$. The accept and the reject vertices are denoted by $\mathsf{BP}[\mathsf{acc}]$ and $\mathsf{BP}[\mathsf{rej}]$ respectively.
- $\mathsf{Layers}(\mathsf{BP})$ denotes the sequence of layers in the branching program $\mathsf{BP}$.
- The length of a branching program is the number of layers in it. That is, length of $\mathsf{BP}$ is $|\mathsf{Layers}(\mathsf{BP})|$.
- We assume that all the layers in a branching program have the same number of vertices [4]. The width of a branching program is then defined to be the number of vertices in a single layer.
- The size of a branching program is the number of vertices in the branching program. It is essentially the product of the width and the number of layers in the branching program.

It is known that every $\mathsf{NC}^1$ circuit can be represented by a polynomial sized formula. Alternately, every $\mathsf{NC}^1$ circuit can also be represented by a layered branching program of size $4^d$, where $d$ is the depth of the circuit[Bar86].

In this work, we try to explore other transformations of formulas to layered branching programs such that the size does not grow exponentially in the depth. For completion of exposition, we describe how we can efficiently transform formulas over AND and NOT gates to layered branching programs satisfying certain conditions. These would then be used in obfuscation later. For a similar transformation for formulas over AND, XOR and NOT gates, refer to [SWW99].

In this work, as in Barak et al. [BGK+14], we would be converting our branching programs to a kind of matrix branching programs that are both oblivious and dual-input. A branching program $\mathsf{BP}$ is said to be input oblivious if its evaluation function depends only on the input length of the function. As a consequence, all the oblivious branching programs corresponding to functions of the same input length have the same evaluation function. In this work, we would be obfuscating "relaxed" matrix branching programs which are dual input and oblivious. We define these in the next section.

## 2.3 Relaxed Matrix Branching Programs

In the previous subsection, we defined layered branching programs using graph theoretic terminology. An alternate way of describing layered branching programs is through matrices. This particular representation of branching programs using matrices is termed as *matrix branching programs* which was first formally defined by [GGH+13b]. We define below the notion of oblivious matrix branching programs.

**Definition 1** (Oblivious Matrix Branching Programs)**.** *[GGH+13b] A branching program of width $w$ and length $n$ for $\ell$-bit inputs is given by a $w \times w$ permutation matrix $P_{\mathsf{reject}}$ such that $P_{\mathsf{reject}} \neq I_{w\times w}$ and by a sequence:*

$$\mathsf{BP} = (\mathsf{inp}, B_{i,0}, B_{i,1})_{i\in[n]},$$

*where $B_{i,b}$, for $i \in [n], b \in \{0,1\}$, are $w \times w$ permutation matrices and $\mathsf{inp} : [n] \to [\ell]$ is the evaluation function of $\mathsf{BP}$. The output of $\mathsf{BP}$ on input $x \in \{0,1\}^\ell$, denoted by $\mathsf{BP}(x)$, is determined as follows:*

[4] This can be ensured by adding dummy vertices in each layer.

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

*Further* $\mathsf{inp} : [n] \to [\ell]$ *is a fixed function independent of the function (with n-bit inputs) being evaluated.*

Barrington [Bar86] showed that every circuit with depth $d$ and fan-in 2 can be represented by a matrix branching program of length at most $4^d$ and width 5. Note that the size of the matrix branching program representation is exponential in the depth and this turns out to be the bottleneck in efficiency when we use the obfuscation scheme [BGK+14] incorporating this representation. In this work, we represent formulae using a variant of matrix branching programs that we term *relaxed matrix branching programs* (RMBPs). Later, we transform a formula into a relaxed matrix branching program whose size is polynomial in the size of formula. The main difference between a matrix branching program and its relaxed version is the following. In a matrix branching program, given the output of the branching program, the product of the matrices is uniquely determined (either $I_{w\times w}$ or $P_{\mathsf{reject}}$). In a relaxed matrix branching program, the output of the branching program does not uniquely determine the product of matrices. Specifically, the output of the branching program is determined only by a single entry in the product of matrices. We formally define the notion of oblivious relaxed matrix branching programs below.

**Definition 2** (Oblivious Relaxed Matrix Branching Programs)**.** *A branching program of size $w$ and length $n$ for $\ell$-bit inputs is given by a sequence:*

$$\mathsf{BP} = (\mathsf{inp}, B_{i,0}, B_{i,1})_{i\in[n]},$$

*where $B_{i,b}$, for $i \in [n], b \in \{0,1\}$, are $w \times w$ full-rank matrices and $\mathsf{inp} : [n] \to [\ell]$ is the evaluation function of $\mathsf{BP}$. The output of $\mathsf{BP}$ on input $x \in \{0,1\}^\ell$, denoted by $\mathsf{BP}(x)$, is determined as follows:*

$$\mathsf{BP}(x) = 1 \text{ if and only if } \Big(\prod_{i=1}^{n} B_{i,x_{\mathsf{inp}(i)}}\Big)[1,w] \neq 0$$

*Further $\mathsf{inp} : [n] \to [\ell]$ is a fixed function independent of function (with n-bit inputs) being evaluated.*

**Dual-input Relaxed Matrix Branching Programs.** We now define the notion of *dual-input relaxed matrix branching programs*. A dual-input relaxed matrix branching program has two evaluation functions $\mathsf{inp}_1$ and $\mathsf{inp}_2$ which are functions from $[n] \to [\ell]$. Consider a layered branching program $\mathsf{BP}$. We associate to layer $i$ matrices $B_{i,b_1,b_2}$ for $i \in [n], b_1, b_2 \in \{0,1\}$ such that,

$$\mathsf{BP}(x) = 1 \text{ if and only if } \Big(\prod_{i=1}^{n} B_{i,x_{\mathsf{inp}_1(i)},x_{\mathsf{inp}_2(i)}}\Big)[1,w] \neq 0$$

In this work, we are interested in dual-input relaxed matrix branching programs that are also oblivious.

### 2.4 "Virtual Black-Box" Obfuscation in an Idealized Model

We now give the definition of virtual black-box obfuscation in the idealized model. Since the model we are working is exactly the same as studied by Barak et al. [BGK+14], we state the definition verbatim from their paper.

**Definition 3** ("Virtual Black-Box" Obfuscation in an $\mathcal{M}$-idealized model)**.** *For a (possibly randomized) oracle $\mathcal{M}$, and a circuit class $\{\mathcal{C}_\ell\}_{\ell\in\mathbb{N}}$, we say that a uniform PPT oracle machine $\mathcal{O}$ is a "Virtual Black-Box" Obfuscator for $\{\mathcal{C}_\ell\}_{\ell\in\mathbb{N}}$ in the $\mathcal{M}$-idealized model, if the following conditions are satisfied:*

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

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

  *where the probability is over the coins of $\mathcal{C}$.*

- *Polynomial Slowdown: there exist a polynomial $p$ such that for every $\ell \in \mathbb{N}$ and every $C \in \mathcal{C}_\ell$, we have that $|\mathcal{O}^{\mathcal{M}}(C)| \leq p(|C|)$.*

- *Virtual Black-Box: for every PPT adversary $\mathcal{A}$ there exist a PPT simulator $\mathsf{Sim}$, and a negligible function $\mu$ such that for all PPT distinguishers $D$, for every $\ell \in \mathbb{N}$ and every $C \in \mathcal{C}_\ell$:*

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

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

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

# 3 Formulae to Relaxed Matrix BPs

The transformation from polynomial sized formulae to branching programs is a well studied problem [Bar86, Cle90, SWW99]. Even though several of the previous works [Cle90, SWW99] deal with obtaining an efficient transformation from formula to branching programs it is not clear whether the resulting branching programs can then be efficiently represented by relaxed matrix branching programs. Indeed this is the reason why we can not use the transformations in the previous works in a black box manner. Yet another way to obtain efficient relaxed matrix branching programs would be to balance the formula first using efficient techniques [BB94, Bre74, BCE91, Spi71] and then apply Barrington's theorem to the balanced formula. But as mentioned in the Introduction (see Table 1), our transformation still beats this approach in efficiency if you start with unbalanced formulae.

Our transformation from a formula to a relaxed matrix branching program can be broken into two steps. In the first step, we transform a formula into layered branching programs. In the second step, we represent these layered branching programs as dual-input oblivious relaxed matrix branching programs.

To make these transformations modular, below we describe two conditions, which if satisfied by the layered branching program, can be used to do obfuscation using our techniques. The conditions (also stated in the introduction) are the following.

- (Layered) The branching program is layered, i.e. the vertices can be partitioned into different layers such that each layer is labeled with a input bit $x_i$. In other words, all the outgoing edges from vertices of this layer are labeled with $x_i$.

- (No back edges) If there is an edge from $j^{th}$ layer to $k^{th}$ layer, then $j < k$.

We call such a branching program a *layered branching program with no back edges*.

In the next we give this transformation for boolean formulas over AND and NOT gates for completion of exposition. A more general transformation was also provided by [SWW99] which works for formulas AND, XOR, and NOT gates.

## 3.1 Formulas to Layered BPs

In this section, we give a transformation of boolean formulas over AND and NOT gates to a layered branching program with no back edges as described above.
Consider a formula, denoted by $\mathsf{F}$. We inductively transform $\mathsf{F}$ to a layered branching program. The base case corresponds to the input wires. Consider an input wire $w$. Let $index(w)$ be $i$. We construct a branching program for $w$ as follows. The branching program, denoted by $\mathsf{BP}_w$ consists of three vertices denoted by $\mathsf{source}$, $\mathsf{acc}$ and $\mathsf{rej}$. There is an edge labeled 0 from $\mathsf{source}$ to the $\mathsf{rej}$ vertex and an edge labeled 1 from $\mathsf{source}$ to $\mathsf{acc}$ (see Figure 1). The evaluation function associated to this wire, denoted by $\mathsf{inp}_w$, is $\mathsf{inp}_w : [1] \to \{i\}$.

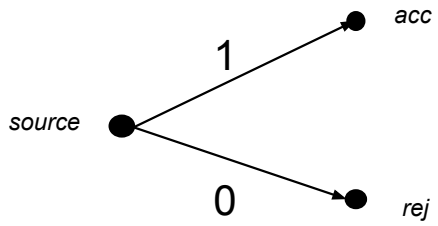


**Figure 1** This denotes the branching program for an input wire.

We proceed to the induction hypothesis. Consider a gate $G$. We have two cases depending on whether it is an AND gate or a NOT gate.

*Case (1)* AND *gate*:- Let $G_1$ and $G_2$ be two gates (or input wires) such that the output wires of $G_1$ and $G_2$ are fed to $G$. Let $\mathsf{BP}_{G_1}$ and $\mathsf{BP}_{G_2}$ be the branching programs associated to gates $G_1$ and $G_2$ respectively. We construct a branching program for $G$, denoted by $\mathsf{BP}_G$, using $\mathsf{BP}_{G_1}$ and $\mathsf{BP}_{G_2}$ in the following steps (see Figure 2) such that the following holds.

- $\mathsf{BP}_{G_1}$ and $\mathsf{BP}_{G_2}$ are subgraphs in $\mathsf{BP}_G$.
- Merge the accept node of $\mathsf{BP}_{G_1}$, which is denoted by $\mathsf{BP}_{G_1}[\mathsf{acc}]$, with the source node of $\mathsf{BP}_{G_2}$, which is $\mathsf{BP}_{G_2}[\mathsf{source}]$. Similarly, merge the reject node of $\mathsf{BP}_{G_1}$, namely $\mathsf{BP}_{G_1}[\mathsf{rej}]$, with the reject node of $\mathsf{BP}_{G_2}$, namely $\mathsf{BP}_{G_2}[\mathsf{rej}]$.
- We have $\mathsf{Layers}(\mathsf{BP}_G) = \Big(\mathsf{Layers}(\mathsf{BP}_{G_1}) \cup \mathsf{Layers}(\mathsf{BP}_{G_2})\Big) \backslash \mathcal{L}$, where $\mathcal{L}$ is a layer consisting of vertices $\{BP_{G_1}[\mathsf{acc}], \mathsf{BP}_{G_1}[\mathsf{rej}]\}$.
- Let $\mathsf{inp}_{G_1} : [t_1] \to [\ell]$ and $\mathsf{inp}_{G_2} : [t_2] \to [\ell]$ be the evaluation functions of $\mathsf{BP}_{G_1}$ and $\mathsf{BP}_{G_2}$ respectively, where $t_1$ and $t_2$ are the number of layers in $\mathsf{BP}_{G_1}$ and $\mathsf{BP}_{G_2}$ respectively. We construct an evaluation function $\mathsf{inp}_G : [t_1 + t_2 - 1] \to [\ell]$ for $\mathsf{BP}_G$ as follows. Set $\mathsf{inp}_G(i)$ to be $\mathsf{inp}_{G_1}(i)$ for all $i \in [t_1 - 1]$ and $\mathsf{inp}_G(t_1 + j - 1) = \mathsf{inp}_{G_2}(j)$ for all $j \in [t_2]$.

Observe that $\mathsf{BP}_G$ is a layered branching program.

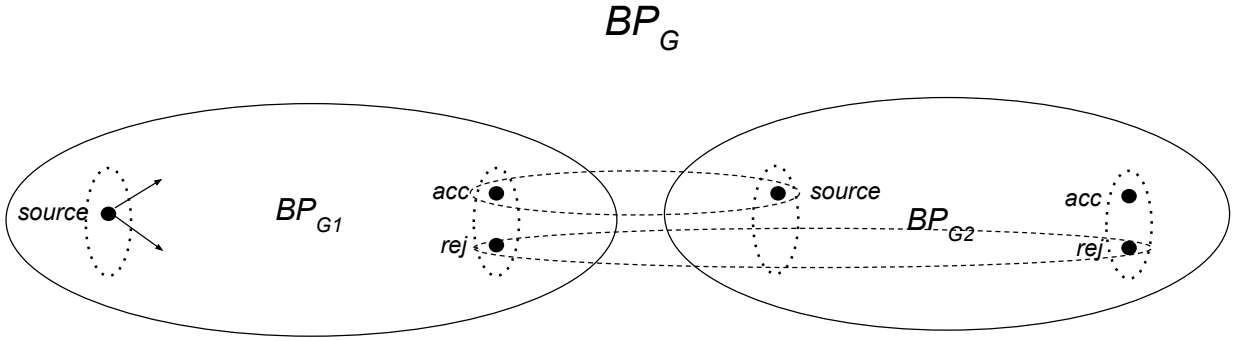


**Figure 2** This denotes the branching program for an $AND$ gate. The accept vertex in $\mathsf{BP}_{G_1}$ is merged with the source vertex in $\mathsf{BP}_{G_2}$. The reject vertex in $\mathsf{BP}_{G_1}$ is merged with the reject vertex in $\mathsf{BP}_{G_2}$.

*Case (2)* NOT *gate*:- Let $G'$ be a gate such that the output wire of $G'$ is fed into the gate $G$. Let $\mathsf{BP}_{G'}$ be a branching program associated to the gate $G'$. We construct a branching program for $G$, denoted by $\mathsf{BP}_G$, using $\mathsf{BP}_{G'}$ in the following steps (see Figure 3).

- The vertex set of $\mathsf{BP}_G$ is the vertex set of $\mathsf{BP}_{G'}$.
- $\mathsf{BP}_G$ is same as the graph $\mathsf{BP}_{G'}$ upto relabeling of the vertices. The accept vertex in $\mathsf{BP}_G$ is the same as $\mathsf{BP}_{G'}[\mathsf{rej}]$ and the reject vertex in $\mathsf{BP}_G$ is the same as $\mathsf{BP}_{G'}[\mathsf{acc}]$.
- We have $\mathsf{Layers}(\mathsf{BP}_G) = \mathsf{Layers}_{\mathsf{BP}_{G'}}$.
- Let $\mathsf{inp}_{G'} : [t'] \to [\ell]$ be the evaluation function for $\mathsf{BP}_{G'}$, where $t'$ is the number of layers in $\mathsf{BP}_{G'}$. We construct an evaluation function $\mathsf{inp}_G : [t'] \to [\ell]$ for $\mathsf{BP}_G$ as follows. First set $\mathsf{inp}_G(i)$ to be $\mathsf{inp}_{G'}(i)$ for all $i \in [t']$.

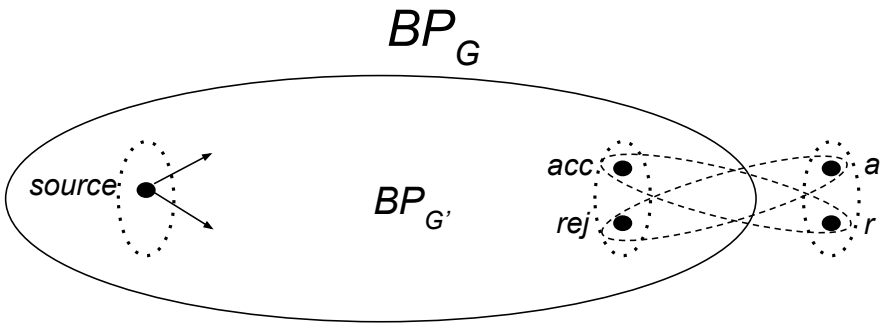


**Figure 3** This denotes the branching program for a $NOT$ gate. Vertices $a$ and $r$ denote the accept and reject vertices of the $\mathsf{BP}_G$ respectively and they are merged with the reject and the accept vertices of $\mathsf{BP}_{G'}$. This effectively amounts to switching of accept and reject vertices in $\mathsf{BP}_{G'}$.

Let the resulting branching program be $\mathsf{BP}_\mathsf{F}$. Denote the layers of $\mathsf{BP}_\mathsf{F}$ to be $\mathsf{Layers}(\mathsf{BP}) = \{\mathcal{L}_1, \ldots, \mathcal{L}_n\}$. Let the size of $\mathsf{BP}_\mathsf{F}$ be $w$. Further, number the vertices in $\mathsf{BP}_\mathsf{F}$ to be $\{1, \ldots, w\}$ such that there does not exist an edge from $i$ to $j$ where $i > j$. This can be achieved using topological sorting (since the branching program is a dag). Further we will achieve this in such a way that the source vertex is numbered 1 and the accept vertex is numbered $w$. Also denote the evaluation function for $\mathsf{BP}_\mathsf{F}$ by $\mathsf{inp}$.

Note that the above construction satisfies the properties of a layered branching program with no back edges. Next, we prove that the above described layered branching program correctly evaluates the formula $\mathsf{F}$.

**Lemma 1.** *For every input $x \in \{0,1\}^l$, we have $\mathsf{F}(x) = 1$ if and only if $\mathsf{BP}_{\mathsf{F}}(x) = 1$.*

*Proof.* We prove this by induction on the number of gates in the formula. The base case consists of just one input wire. In this case, the lemma follows from the base case in our construction. We will now proceed to the induction hypothesis. We will assume that the lemma is true for formulas with $t-1$ gates. We now show that the lemma is true for formulas with $t$ gates. Consider a formula $\mathsf{F}_G$ having $t$ gates with the output gate of this formula being $G$. We now argue that the lemma is true for $\mathsf{BP}_G|_x$, which is the branching program associated to the gate $G$. We consider two cases depending on the type of the gate $G$.

*Case (i) AND gate*:- Let $G_1$ and $G_2$ be two gates such that the output of these gates are input to $G$. We denote the formula associated to $G_1$ (resp., $G_2$) to be $\mathsf{F}_{G_1}$ (resp., $\mathsf{F}_{G_2}$). Since $\mathsf{F}_{G_1}$ (resp., $\mathsf{F}_{G_2}$) has at most $t-1$ gates, we can apply the induction hypothesis. That is, the branching program $\mathsf{BP}_{G_1}$ (resp., $\mathsf{BP}_{G_2}$) constructed from $\mathsf{F}_{G_1}$ (resp., $\mathsf{F}_{G_2}$) is such that there exists a path from the source to the accept vertex and no path exists from the source to the reject vertex in $\mathsf{BP}_{G_1}|_x$ (resp., $\mathsf{BP}_{G_2}|_x$) iff $\mathsf{F}_{G_1}(x) = 1$ (resp., $\mathsf{F}_{G_2}(x) = 1$).

Consider the case when $\mathsf{F}_G(x) = 1$. In this case, both $\mathsf{F}_{G_1}(x) = 1$ and $\mathsf{F}_{G_2}(x) = 1$. We claim that there exists a path from the source vertex of $\mathsf{BP}_G|_x$ to its accept vertex. This is because, there exists a path from the source vertex of $\mathsf{BP}_{G_1}|_x$ (which is also the source vertex in $\mathsf{BP}_G|_x$) to its accept vertex, which is merged with the source vertex of $\mathsf{BP}_{G_2}|_x$. Further, there is a path from the source vertex of $\mathsf{BP}_{G_2}|_x$ to the accept vertex of $\mathsf{BP}_{G_2}|_x$ (which is also an accept vertex in $\mathsf{BP}_G|_x$). We now argue that there does not exist any path from the source to the reject vertex in $\mathsf{BP}_G|_x$. Any path from the source vertex in $\mathsf{BP}_G|_x$ to its reject vertex should either contain a path from the source vertex in $\mathsf{BP}_{G_1}|_x$ to its reject vertex or a path from the source vertex in $\mathsf{BP}_{G_2}|_x$ to its reject vertex. By induction hypothesis, the paths in both $\mathsf{BP}_{G_1}|_x$ and $\mathsf{BP}_{G_2}|_x$ cannot exist and hence there is no path from the source vertex in $\mathsf{BP}_G|_x$ to its reject vertex.

We now consider the case when $\mathsf{F}_G(x) = 0$. This means that either $\mathsf{F}_{G_1}(x) = 0$ or $\mathsf{F}_{G_2}(x) = 0$. We now claim that there does not exist a path from the source vertex in $\mathsf{BP}_G(x)|_x$ to its accept vertex. Crucial to proving this is the observation that every path from the source vertex of $\mathsf{BP}_G|_x$ to its accept vertex needs to pass through the accept vertex of $\mathsf{BP}_{G_1}|_x$ (which is the source vertex of $\mathsf{BP}_{G_2}|_x$). Without loss of generality let $\mathsf{F}_{G_1}(x)$ be 0. Then there is no path from the source vertex of $\mathsf{BP}_{G_1}|_x$ to its accept vertex and from the previous observation, there cannot be any path from the source vertex of $\mathsf{BP}_G|_x$ to its accept vertex. Similarly when $\mathsf{F}_{G_2}(x) = 0$ then there cannot be any path from the source vertex of $\mathsf{BP}_{G_2}|_x$ to its accept vertex. We now show that there exists a path from the source vertex in $\mathsf{BP}_G|_x$ to its reject vertex. Let $\mathsf{F}_{G_1}(x) = 0$ then there is a path from the source vertex of $\mathsf{BP}_{G_1}|_x$ to its reject vertex, which is the same as the reject vertex of $\mathsf{BP}_G|_x$. When $\mathsf{F}_{G_2}(x) = 0$ and $\mathsf{F}_{G_1}(x) = 1$ then there is a path from the source vertex in $\mathsf{BP}_{G_1}|_x$ to its accept vertex which is merged with the source vertex of $\mathsf{BP}_{G_2}|_x$. Further there is a path from the source vertex in $\mathsf{BP}_{G_2}|_x$ to its reject vertex which is nothing but the reject vertex in $\mathsf{BP}_G|_x$. This completes this case.

*Case (ii) NOT gate*:- Let $G_1$ be a gate such that the output of the gate is input to $G$. We denote the formula associated to $G_1$ to be $\mathsf{F}_{G_1}$. Since $\mathsf{F}_{G_1}$ has at most $t-1$ gates, we can apply the induction hypothesis. That is, the branching program $\mathsf{BP}_{G_1}$ constructed from $\mathsf{F}_{G_1}$ is such that there exists a path from the source to the accept vertex in $\mathsf{BP}_{G_1}|_x$ iff $\mathsf{F}_{G_1}x = 1$.

Consider the case when $\mathsf{F}_G(x) = 1$. In this case, $\mathsf{F}_{G_1}(x) = 0$. We claim that there exists a path from the source vertex of $\mathsf{BP}_G|_x$ to its accept vertex. From the induction hypothesis, there is a path from the source vertex to the reject vertex in $\mathsf{BP}_{G_1}|_x$. Since the source vertex of

$\mathsf{BP}_{G_1}|_x$ is the same as the source vertex of $\mathsf{BP}_G|_x$ and the reject vertex of $\mathsf{BP}_{G_1}|_x$ is the same as the accept vertex of $\mathsf{BP}_G|_x$, our claim follows. We now argue that there is no path from the source vertex in $\mathsf{BP}_G|_x$ to its reject vertex. This follows from the fact that no path exists from the source vertex in $\mathsf{BP}_{G_1}|_x$ to its accept vertex and also the accept vertex in $\mathsf{BP}_{G_1}|_x$ is same as the reject vertex in $\mathsf{BP}_G|_x$.

The case when $\mathsf{F}_G(x) = 0$ symmetrically follows from the case when $\mathsf{F}_G(x) = 1$. This completes the proof of this case as well as the lemma. □

**Analysis of the size of $\mathsf{BP}_\mathsf{F}$.** Now we will prove a bound on the size of the branching program obtained by the above described procedure in terms of the size of the formula.

**Theorem 1.** *Let $s$ be the size of the formula. Then the size of the branching program $\mathsf{BP}_\mathsf{F}$ (obtained using the transformation in Section 3.1) is at most $2s + 4$.*

*Proof.* We prove this by induction on the size of the formula. Base case is $s = 1$, i.e. the formula just consists of an input wire. In this case, the size of the branching program is 3 and this is clearly less than $2s + 4$. Now we argue about the size of the branching program for a formula of size $s$ assuming that the theorem holds for all formulas of size at most $(s - 1)$. Let the output gate in the formula be gate $G$. We do a case analysis on $G$.

- $G$ is an AND gate. Let $\mathsf{F}_1$ and $\mathsf{F}_2$ be two formulas such that the output wires of both of these are fed into $G$. Let the size of the formula $\mathsf{F}_1$ (resp., $\mathsf{F}_2$) be $s_1$ (resp., $s_2$). By induction hypothesis, we have the size of the branching program associated to $\mathsf{F}_1$ (resp., $\mathsf{F}_2$) is at most $2s_1 + 4$ (resp., $2s_2 + 4$). The size of the branching program $\mathsf{BP}_\mathsf{F}$ is given as follows.

$$\begin{aligned} |\mathsf{BP}_\mathsf{F}| &= |\mathsf{BP}_{\mathsf{F}_1}| + |\mathsf{BP}_{\mathsf{F}_2}| - 2 \\ &\leq 2s_1 + 4 + 2s_2 + 4 - 2 \\ &\leq 2(s_1 + s_2 + 1) + 4 \\ &= 2s + 4, \end{aligned}$$

  where the last equality holds by the fact that $s = s_1 + s_2 + 1$.

- $G$ is a NOT gate. Let $F'$ be the formula such that output wire of $F'$ is fed into $G$. Let the size of $F'$ be $s'$. By induction hypothesis, we have the size of the branching program associated to $F'$ be at most $2s' + 4$. The size of the branching program $\mathsf{BP}_\mathsf{F}$ is given as follows.

$$\begin{aligned} |\mathsf{BP}_\mathsf{F}| &= |\mathsf{BP}_{F'}| \\ &\leq 2s' + 4 \\ &\leq 2(s' + 1) + 4 \\ &= 2s + 4, \end{aligned}$$

  where the last equality holds by the fact that $s = s' + 1$.

□

**Corollary 1.** *The number of layers in the branching program $\mathsf{BP}_\mathsf{F}$ is at most $s + 2$.*

*Proof Sketch.* This follows by observing that each layer in the branching program has two vertices.

**Theorem 2** ([SWW99])**.** *Given a boolean formula of size $s$ over* AND, NOT, AND XOR*, there exists a layered branching program with no back edges with $O(s)$ layers and $O(s^{1.195})$ vertices.*

Note that the branching program given by [SWW99] satisfies the properties listed before and hence our techniques described in the subsequent sections are applicable to it.

### 3.2 Representing Layered BPs as Relaxed Matrix BPs

Recall that as mentioned before, in order to adapt the techniques of obfuscation from [BGK+14], we need to convert our formula into a relaxed matrix branching program. In the previous section we described how to convert a formula into a graph branching program. In this section, we describe how to take the graph branching program $\mathsf{BP_F}$ obtained above and convert it to a set of matrices. Let $w$ denote the size of the graph branching program, i.e. there are $w$ vertices in the graph representation of $F$. Let $n$ be the number of layers in $\mathsf{BP_F}$. Denote the layers in $\mathsf{BP_F}$ by $\mathcal{L}_1, \ldots, \mathcal{L}_n$ excluding the last layer which does not have any outgoing edges. Now we will construct a length $n$ relaxed matrix branching program such that all the matrices will have full rank. The evaluation function for the relaxed matrix branching program is the same as the graph branching program. Then we will prove some properties about these matrices.

Corresponding to each layer in the branching program, we will have two $(w \times w)$ matrices $B_{i,0}$ and $B_{i,1}$ as follows:

$B_{i,0}$ : For every edge $e = (u, v)$ in $\mathsf{BP_F}$, where $u \in \mathcal{L}_i$ and $e$ is labeled to be 0, set the entry $B_{i,0}[u, v]$ to be 1. For each $u \in [w]$, set $B_{i,0}[u, u] = 1$. Set the rest of the entries in $B_{i,0}$ to be 0.

$B_{i,1}$ : For every edge $e = (u, v)$ in $\mathsf{BP_F}$, with $u \in \mathcal{L}_i$ and $e$ is labeled 1, set the entry $B_{i,1}[u, v]$ to be 1. For each $u \in [w]$, set $B_{i,1}[u, u] = 1$. Rest of the entries in $B_{i,1}$ are set to be 0.

**Claim 1.** *Matrices $B_{i,b}$ are full rank for all $i \in n$, $b \in \{0, 1\}$.*

*Proof Sketch.* Each $B_{i,b}$ is upper triangular because there does not exist any edge from $i$ to $j$ for $i > j$ and also all diagonal entries set to 1.

Looking ahead, we will use this claim to randomize the relaxed matrix branching program. In our proof of security we will need show that these randomized matrices can be simulated. Here, we will crucially rely on the fact that the matrices are full rank.

**Lemma 2.** *Consider an input $x \in \{0, 1\}^l$. Denote the product $\prod_{i=1}^{n} B_{i,x_{\mathsf{inp}(i)}}$ by $P$. Then, $P[1, w] \geq 1$ if and only if $\mathsf{BP_F}(x) = 1$.*

*Proof.* We argue this by induction. We define graph $G_j$, for $j \in [n]$, to be a subgraph of $\mathsf{BP}$ as follows. It consists of all the vertices in the layers $\mathcal{L}_1, \ldots, \mathcal{L}_{j+1}$ and any two vertices in $G_j$ have an edge if and only if the correpsonding two vertices in $\mathsf{BP}|_x$ have an edge. We will denote the vertex set associated to $G_j$ as $V_j$. Without loss of generality we will assume that $V_j = \{1, \ldots, 2(j + 1)\}$, since each layer has two vertices.

At each point in the induction we maintain the invariant that $P_j[u, v] = c_{u,v}$, where $P_j = \prod_{i=1}^{j} B_{i,x_{\mathsf{inp}(i)}}$ and $u, v \in V_j$ and $c_{u,v}$ is the number of possible paths from $u$ to $v$ in $G_j$.

The base case in the induction step is for the case of $G_1$ and the invariant follows from the definition of $G_1$. We now proceed to the induction hypothesis. Assume that the matrices $(B_{i,x_{\mathsf{inp}(i)}})_{i \in [j]}$, for $j < n$ is such that their product, which is $P_j$, satisfies the condition that $P_j[u, v]$ is the number of paths from $u$ to $v$ in graph $G_j$. Consider the product $P_{j+1} = \prod_{i=1}^{j+1}(B_{i,x_{\mathsf{inp}(i)}})$ which is essentially the product $P_j \cdot B_{j+1,x_{\mathsf{inp}(j+1)}}$. Now, consider $P_{j+1}[u, v] = \sum_{i=1}^{w} P_j[u, i] B_{j+1,x_{\mathsf{inp}(j+1)}}[i, v]$ for $u, v \in V_{j+1}$. Each term indicates the total number of paths from $u$ to $v$ with $i$ as an intermediate vertex in the graph $G_{j+1}$. Note that an intermediate vertex of any path of length at least 2 in $G_{j+1}$ should be in $V_j$. And the summation of all these terms indicates the total number of paths from $u$ to $v$ in $G_{j+1}$.

We have established that $P_n[u, v]$ represents the number of paths from the $u$ to $v$ in graph $G_n$. But $G_n$ is nothing but the graph $\mathsf{BP}|_x$ and $P_n$ is nothing but the matrix $P$. This shows that $P[u, v]$ denotes the number of paths from $u$ to $v$ in graph $\mathsf{BP}|_x$ and more specifically, $P[1, w]$ is non-zero iff $\mathsf{BP_F}(x) = 1$. This proves the lemma. □

The proof of the following corollary follows directly from Lemma 1 and Lemma 2.

**Corollary 2.** *Consider an input $x \in \{0,1\}^\ell$. Denote the product $\prod_{i=1}^{n} B_{i,x_{\mathsf{inp}(i)}}$ by $P$. Then, $P[1,w] \geq 1$ if and only if $\mathsf{F}(x) = 1$.*

**Dual-Input Oblivious RMBPs.** We further make our relaxed matrix branching program oblivious and dual-input. As a first step, we will make our relaxed matrix branching program oblivious, i.e. make the evaluation function $\mathsf{inp}$ independent of the formula $F$. Wlog, assume that the length of the relaxed matrix branching program, $n$, is a multiple of $(\ell-1)$, i.e. $n = k \cdot (\ell-1)$ for some $k \in \mathbb{N}$. If this not the case, add at most $(\ell-2)$ pairs of identity matrices of dimension $w \times w$ to the relaxed matrix branching program. We will use this assumption while making the branching program dual input. Now we will describe a new (relaxed) matrix branching program of length $n' = n \cdot \ell$ and width $w$ and evaluation function $\mathsf{inp}_1$ as follows:

- Define $\mathsf{inp}_1(i) = i \bmod \ell$ for all $i \in [n]$.
- For each $j \in [n]$, $M_{(j-1)\cdot\ell+\mathsf{inp}(j),b} = B_{j,b}$ for $b \in \{0,1\}$. Rest all matrices are set to $I_{w\times w}$.

Informally the above transformation can be described as follows: In $j^{th}$ block of $\ell$ matrices, all the matrices are identity matrices apart from the matrices at index $\mathsf{inp}(j)$. At this index, we place the two non-trivial matrices $B_{j,0}$ and $B_{j,1}$ which help in actual computation.

**Claim 2.** *For any input $x \in \{0,1\}^\ell$, $\prod_{i=1}^{n} B_{i,x_{\mathsf{inp}(i)}} = \prod_{i=1}^{n'} M_{i,x_{\mathsf{inp}_1(i)}}$.*

Now we make the above relaxed matrix branching program dual-input, by pairing the input position used at each index with a dummy input position in an oblivious manner which is independent of the formula. For convenience of notation, we will also ensure that each pair of input bits is used as the selector same number of times. We will ensure that at any index of the RMBP, the two input positions used are distinct, i.e. $\mathsf{inp}_1(i) \neq \mathsf{inp}(i)$ for any $i \in [\ell]$. We define the evaluation function $\mathsf{inp}_2$ as follows: Consider $i$ of the form $k_1\ell(\ell-1) + k_2\ell + k_3$ then

$$\mathsf{inp}_2(i) = ((k_2 + k_3) \bmod \ell) + 1$$

**Theorem 3.** *For any formula $F$ of size $s$ taking inputs of length $\ell$, there exists an oblivious dual input relaxed matrix branching program $\mathsf{BP} = (\mathsf{inp}_1, \mathsf{inp}_2, \{B_{i,b_1,b_2}\}_{i\in[n],b_1,b_2\in\{0,1\}})$ with length $n \leq ((s+2)+(\ell-2)) \cdot \ell$ and width $w \leq 2s+4$.*

**Remark 1.** *Note that for obfuscation we require that the length and the width of the relaxed matrix branching program to be determined by the size of the formula itself. Hence, we will use $n = ((s+2)+k) \cdot \ell$ such that $k \leq \ell - 2$ and $(s+2+k)$ is a multiple of $\ell-1$ for the analysis above to go through, and $w = 2s+4$. We will obtain this by padding the RMBP obtained with identity matrices.*

# 4 Randomization of Relaxed Matrix BPs

In this section, we describe how to randomize the matrices in the relaxed matrix branching program described in Section 3.1. We will do this in two steps via procedures $\mathsf{randBP}$ and $\mathsf{randBP}'$.

Though the procedure $\mathsf{randBP}$ to randomize our matrices is very similar to the Kilian's randomization (also used in [GGH+13b, BGK+14]), the way we will simulate these matrices will deviate from that of Kilian. This is because in standard matrix branching programs, depending on the output of the function on $x$, branching program evaluates to one of the two fixed matrices: $P_{\mathsf{accept}}$ or $P_{\mathsf{reject}}$. In our case, we have a relaxed notion of the matrix branching programs, in which we can only fix of entry of the final matrix (see Lemma 2).

**Notation.** We will denote the relaxed matrix branching program as $\mathsf{BP} = (\mathsf{inp}_1, \mathsf{inp}_2, \{B_{i,b_1,b_2}\}_{i\in[n],b_1,b_2\in\{0,1\}})$ with length $n$, width $w$ and input of $\ell$ bits. For any $x \in \{0,1\}^\ell$, define $P_x := \prod_{i=1}^{n} B_{i,x_{\mathsf{inp}_1(i)},x_{\mathsf{inp}_2(i)}}$ and $\mathsf{BP}\Big|_x := (\{B_{i,x_{\mathsf{inp}_1(i)},x_{\mathsf{inp}_2(i)}}\}_{i\in[n]})$. Let $e_1, e_w \in \{0,1\}^w$ be such that $e_1 = (1,0,0,\ldots,0)$ and $e_w = (0,0,\ldots,0,1)$. For notational convience, let $e_1$ be a row vector and $e_w$ we a column vector.

**Procedure $\mathsf{randBP}$.** The input to the randomization procedure is an oblivious dual-input RMBP $\mathsf{BP} = (\mathsf{inp}_1, \mathsf{inp}_2, \{B_{i,b_1,b_2}\}_{i\in[n],b_1,b_2\in\{0,1\}})$ of length $n$, width $w$ and input of $\ell$ bits.

$\mathsf{randBP}(\mathsf{BP})$:

- Pick a prime $p$ of length $O(n)$.
- Pick $n+1$ random full-rank matrices $R_0, \ldots, R_n \in \mathbb{Z}_p^{w\times w}$.
- Compute the matrices $\tilde{B}_{i,b_1,b_2} = R_{i-1} \cdot B_{i,b_1,b_2} \cdot R_i^{-1}$ for all $i \in [n]$ and $b_1, b_2 \in \{0,1\}$.
- Finally, compute $\tilde{s} = e_1 \cdot R_0^{-1}$ and $\tilde{t} = R_n \cdot e_w$.
- Output $\widetilde{\mathsf{BP}} = \left(\tilde{s}, \left\{\tilde{B}_{i,b_1,b_2}\right\}_{i\in[n],b_1,b_2\in\{0,1\}}, \tilde{t}\right)$.

**Claim 3.** *For inputs $x \in \{0,1\}^\ell$, following holds*

$$\tilde{s} \cdot \prod_{i=1}^{n} \tilde{B}_{i,x_{\mathsf{inp}_1(i)},x_{\mathsf{inp}_2(i)}} \cdot \tilde{t} = \left(\prod_{i=1}^{n} B_{i,x_{\mathsf{inp}_1(i)},x_{\mathsf{inp}_2(i)}}\right)[1,w].$$

*Proof.* Claim holds as follows:

$$\tilde{s} \cdot \prod_{i=1}^{n} \tilde{B}_{i,x_{\mathsf{inp}_1(i)},x_{\mathsf{inp}_2(i)}} \cdot \tilde{t} = \tilde{s} \cdot R_0 \cdot \prod_{i=1}^{n} B_{i,x_{\mathsf{inp}_1(i)},x_{\mathsf{inp}_2(i)}} \cdot R_n^{-1} \cdot \tilde{t} = \left(\prod_{i=1}^{n} B_{i,x_{\mathsf{inp}_1(i)},x_{\mathsf{inp}_2(i)}}\right)[1,w],$$

where the last equality holds by using $\tilde{s} = e_1 \cdot R_0^{-1}$ and $\tilde{t} = R_n \cdot e_w$. □

**Simulator $\mathsf{Sim}_{\mathsf{BP}}$ for $\mathsf{randBP}$.** Next, we describe the simulator $\mathsf{Sim}_{\mathsf{BP}}$ which simulates the output of $\mathsf{randBP}$ for any input $x$. More formally, let $\mathsf{randBP}(\mathsf{BP})\Big|_x$ be defined as $(\tilde{s}, \{\tilde{B}_{i,x_{\mathsf{inp}_1(i)},x_{\mathsf{inp}_2(i)}}\}_{i\in[n]}, \tilde{t})$. We describe a simulator $\mathsf{Sim}_{\mathsf{BP}}$ which takes as input $(1^s, P_x[1,w])$ and outputs a tuple which is identically distributed to $\mathsf{randBP}(\mathsf{BP})\Big|_x$. Recall that $s$ is the size of the formula.

Before we describe $\mathsf{Sim}_{\mathsf{BP}}$ we will first recall the following theorem. We use the simulator used in the following theorem in our construction of $\mathsf{Sim}_{\mathsf{BP}}$. Like Barak et al. [BGK+14], we adapt the theorem for dual-input RMBPs.

**Theorem 4.** *([Kil88]) Consider a dual-input branching program $\mathsf{BP} = \left\{\mathsf{inp}_1, \mathsf{inp}_2, \{B_{i,x_{\mathsf{inp}_1(i)},x_{\mathsf{inp}_2(i)}}\}_{i\in[n]}\right\}$. There exists a PPT simulator $\mathsf{Sim}_{\mathsf{K}}$ such that for every $x \in \{0,1\}^l$,*

$$\{R_0, \{R_{i-1}B_{i,x_{\mathsf{inp}_1(i)},x_{\mathsf{inp}_2(i)}}R_i^{-1}\}_{i\in[n]}, R_n\} \equiv \mathsf{Sim}_{\mathsf{K}}(1^s, \mathsf{BP}(x))$$

We are now ready to describe $\mathsf{Sim}_{\mathsf{BP}}$.

$\mathsf{Sim}_{\mathsf{BP}}(1^s, P_x[1,w])$:

- Compute the length $n$ and width $w$ of the oblivious dual input RMBP for any formula of size $s$ using Theorem 3 (see Remark 1).

- If $P_x[1,w] \neq 0$, define the matrix $A$ as $A := P_x[1,n] \cdot I_{w\times w}$. Else, $A :=$ "mirror-image" of $I_{w\times w}$ [5].
- Run $\mathsf{Sim}_{\mathsf{K}}(1^n, A)$ to obtain full-rank matrices $R_0, R_1, \ldots, R_{n+1} \in \mathbb{Z}_p^{w\times w}$ such that $\prod_{i\geq 0} R_i = A$. Note that $\mathsf{Sim}_{\mathsf{K}}$ is the simulator for Kilian's randomization as defined in Theorem 4.
- Let $\hat{R}_0 = e_1 \cdot R_0^{-1}$ and $\hat{R}_{n+1} = R_{n+1} \cdot e_w$.
- Output $(\hat{R}_0, R_1, \ldots, R_n, \hat{R}_{n+1})$.

We will prove the following theorem.

**Theorem 5.** *Consider an oblivious dual-input RMBP* $\mathsf{BP} = (\mathsf{inp}_1, \mathsf{inp}_2, \{B_{i,b_1,b_2}\}_{i\in[n],b_1,b_2\in\{0,1\}})$ *of length* $n$*, width* $w$ *and input of* $\ell$ *bits. Then for every* $x \in \{0,1\}^\ell$*,*

$$\left\{ \mathsf{randBP}(\mathsf{BP})\Big|_x \right\} \equiv \left\{ \mathsf{Sim}_{\mathsf{BP}}(1^s, P_x[1,w]) \right\}.$$

Before we prove the theorem, we first state the following lemma from Cramer et al. [CFIK03] that will be useful to prove the theorem. We describe the proof for completeness sake.

**Lemma 3.** *For any* $x, y \in \mathbb{Z}_p^w \backslash \{0\}$ *and a full rank matrix* $M \in \mathbb{Z}_p^{w\times w}$ *there exist full rank matrices* $X, Y \in (\mathbb{Z}_p)^{n\times n}$ *such that the first row of* $X$ *is* $x^T$*, the first column of* $Y$ *is* $y$*, and* $XMY$ *depends only on* $x^T My$*.*

*Proof.* We will consider two cases depending on whether $x^T My$ is either zero or non-zero. When $x^T My \neq 0$, we will find $X$ and $Y$ such that $XMY = (x^T My) \cdot I_{w\times w}$, else we will find $X$ and $Y$ such that $XMY =$ "mirror-image" of $I_{w\times w}$.

*Case (i).* $x^T My \neq 0$: Let $x^T My$ be denoted by $c$. We will now construct a matrix $X$ such that the first row of $X$ is $x^T$. Further, $X$ needs to satisfy the condition that $XMy$ is $ce_j$. To achieve this, we will fix the first row of $X$ to be $x^T$ and then we will pick the remaining rows such that it forms a basis for the space which is orthogonal to $My$. Since, $x^T y \neq 0$, we have the fact that $x^T$ is not in this space and hence $X$ is a full rank matrix. We will construct $Y$ as follows. We will fix the first column of $Y$ to be $y$ and then we will generate the rest of the columns of $Y$ so that $Y = M^{-1}X^{-1}(c \cdot I)$. Since $X$ and $M$ are full-rank matrices, inverses are well-defined. From this, it follows that $XMY = c \cdot I$.

*Case (ii).* $x^T My = 0$: Unlike the previous case, here $x^T$ is in the space orthogonal to $My$. We construct $X$ as follows. We fix the first row of $X$ to be $x^T$. The remaining $w-2$ rows of $X$ are picked so that with $x$ they form a basis of the space orthogonal to $My$. We now pick the last row of $X$ to be such that $XMy = 1$. Observe that $X$ is a full rank matrix. We will now construct $Y$ as follows. The first column of $Y$ is fixed to be $y$ and then we pick the rest of the columns of $Y$ such that $Y = M^{-1}X^{-1}J$, where $J$ is the "mirror-image" of $I$. □

We will denote the procedure described above by $\mathsf{Extend}$. In more detail, $\mathsf{Extend}$ takes as input $(x^T My, x, y, M)$, where $x, y$ and $M$ are as defined in the above lemma and outputs $X$ and $Y$ such that $XMY$ is $(x^T My) \cdot I_{w\times w}$ if $x^T My \neq 0$ else it is "mirror-image" of $I$.

We will now prove Theorem 5.

*Proof of Theorem 5.* In order to prove the theorem, we will define a sequence of hybrids such that the first hybrid is the real experiment (which is $\mathsf{randBP}$) while the last hybrid is the simulated experiment (which is $\mathsf{Sim}_{\mathsf{BP}}$). We will then show that the output distribution of each

[5] The "mirror-image" of $I_{w\times w}$ is a $w \times w$ matrix with 1's only in entries of the form $[i, w-i+1]$ for $i \in [w]$.

hybrid is identical to the output distribution of the previous hybrid which will prove the theorem.

$\underline{\mathsf{Hybrid}_0}$: This is the same as the real experiment. That is, on input $\mathsf{BP}$ and $x$ it first executes $\mathsf{randBP}(\mathsf{BP})$ to obtain $\widetilde{\mathsf{BP}}$. It then outputs $\widetilde{\mathsf{BP}}\Big|_x = (\tilde{s}, \{\tilde{B}_{i,x_{\mathsf{inp}_1(i)},x_{\mathsf{inp}_2(i)}}\}_{i\in[n]}, \tilde{t})$

$\underline{\mathsf{Hybrid}_1}$: We will describe $\mathsf{Hybrid}_1$ as follows. The input to $\mathsf{Hybrid}_1$ is $\widehat{\mathsf{BP}} = \mathsf{BP}\big|_x = (\{B_{i,x_{\mathsf{inp}_1(i)},x_{\mathsf{inp}_2(i)}}\})$. Let $M_i = B_{i,x_{\mathsf{inp}_1(i)},x_{\mathsf{inp}_2(i)}}$.

$\mathsf{Hybrid}_1\Big(\widehat{\mathsf{BP}} = (M_1, \ldots, M_n)\Big)$ :

- Pick a prime $p$ of length $O(n)$.
- Pick $n+1$ random full-rank matrices $R_0, \ldots, R_n \in \mathbb{Z}_p^{w\times w}$.
- Compute the matrices $\tilde{M}_i = R_{i-1} \cdot M_i \cdot R_i^{-1}$ for $i \in [n]$.
- Finally, compute $\tilde{s} = e_1 \cdot R_0^{-1}$ and $\tilde{t} = R_n \cdot e_w$.
- Output $\Big(\tilde{s}, \{\tilde{M}_i\}_{i\in[n]}, \tilde{t}\Big)$.

It can be seen that the output distribution of this hybrid is identical to the output distribution of the previous hybrid $\mathsf{Hybrid}_0$. This is because, picking the matrices corresponding to the input $x$ and then randomizing it (as in $\mathsf{Hybrid}_1$) yields the same distribution as first randomizing the RMBP and later picking the matrices corresponding to the input $x$. Note that for any $i \in [n]$ all $B_{i,b_1,b_2}$, where $b_1, b_2 \in \{0,1\}$ are randomized in the same way using $R_{i-1}$ and $R_i$.

$\underline{\mathsf{Hybrid}_2}$: $\mathsf{Hybrid}_2$ is same as $\mathsf{Hybrid}_1$ except the way we compute $\tilde{s}$ and $\tilde{t}$. The input to $\mathsf{Hybrid}_2$, like the previous hybrid, is $\widehat{\mathsf{BP}} = \mathsf{BP}\big|_x$.

$\mathsf{Hybrid}_2\Big(\widehat{\mathsf{BP}} = (M_1, \ldots, M_n)\Big)$ :

- Pick a prime $p$ of length $O(n)$.
- Pick $n+1$ random full-rank matrices $R_0, \ldots, R_n \in \mathbb{Z}_p^{w\times w}$.
- Compute the matrices $\tilde{M}_i = R_{i-1} \cdot M_i \cdot R_i^{-1}$ for $i \in [n]$.
- Define $P := \prod_{i=1}^{n} M_i$ and $c := e_1 \cdot P \cdot e_w$.
- Execute $\mathsf{Extend}$ on input $(c, e_1, e_w, P)$ to obtain $w \times w$ matrices $S$ and $T$ as described in Lemma 3. Compute $\hat{S} = SR_0^{-1}$ and $\hat{T} = R_n T$. Finally, compute $\tilde{s} = e_1\hat{S}$ and $\tilde{t} = \hat{T}e_w$.
- Output $(\tilde{s}, \{\tilde{M}_i\}_{i\in[n]}, \tilde{t})$.

We claim that this is identical to the previous hybrid. $\mathsf{Hybrid}_1$ and $\mathsf{Hybrid}_2$ differ only in the way $\tilde{s}$ and $\tilde{t}$ are computed. Hence, all we need to show is that the value of $\tilde{s}$ and $\tilde{t}$ is identical in both the hybrids.

In $\mathsf{Hybrid}_2$, $\tilde{s} = e_1\hat{S} = e_1 \cdot (SR_0^{-1}) = (e_1 \cdot S) \cdot R_0^{-1} = x^T \cdot R_0^{-1}$, where $x$ is the first row of $S$. But the first row of $S$ is *eone* and hence, $\tilde{s} = e_1 \cdot R_0^{-1}$, which is same as the value in $\mathsf{Hybrid}_1$. Similarly, we can show this for $\tilde{t}$.

*Observation:* Before we describe $\mathsf{Hybrid}_3$, note that in $\mathsf{Hybrid}_2$, $c = P[1, w]$. Then it follows from Lemma 3 that if $c \neq 0$, $S \cdot P \cdot T = c \cdot I$, else $S \cdot P \cdot T = J$, where $J$ is the "mirror-image" of $I$.

$\underline{\mathsf{Hybrid}_3}$: This is same as the simulated experiment. That is, it takes as input $1^s$ and $P_x[1, w]$ and then executes $\mathsf{Sim}_{\mathsf{BP}}(1^s, P_x[1, w])$. The output of $\mathsf{Hybrid}_3$ is the output of $\mathsf{Sim}_{\mathsf{BP}}$.

**Lemma 4.** *The output distribution of* $\mathsf{Hybrid}_2$ *is identical to the output distribution of* $\mathsf{Hybrid}_3$.

*Proof.* In order to prove the claim, we use the Kilian's theorem for simulating randomized RMBPs (see Theorem 4). We first observe that both $\mathsf{Hybrid}_2$ and $\mathsf{Sim}_{\mathsf{BP}}$ execute the following two steps. They compute $n+2$ matrices and then have a common step where both of them truncate the first and the last matrix. In $\mathsf{Hybrid}_2$ this sequence of matrices is $(SR_0^{-1}, \{R_{i-1}M_iR_i^{-1}\}, R_nT)$. In $\mathsf{Hybrid}_3$, this sequence of matrices is $\mathsf{Sim}_{\mathsf{K}}(1^n, A)$ with $A = c \cdot I_{w\times w}$ if $P_x[1,w] = c \neq 0$ else $A$ is a "mirror-image" of $I_{w\times w}$. So it suffices to show that these distributions are identical. That is,

$$(SR_0^{-1}, \{R_{i-1}M_iR_i^{-1}\}, R_nT) \equiv \mathsf{Sim}_{\mathsf{K}}(1^n, A)$$

This in turn follows directly from Theorem 4. □

This shows that the output distribution of $\mathsf{Hybrid}_0$ is identically distributed to $\mathsf{Hybrid}_3$. This completes the proof of Theorem 5.

**Proceduce $\mathsf{randBP}'$.** We now describe how to further randomize the output of $\mathsf{randBP}$ and then show to simulate this having just the output of $\mathsf{BP}$. The input to $\mathsf{randBP}'$ is randomized relaxed matrix branching program $\widetilde{\mathsf{BP}} = (\tilde{s}, \{\tilde{B}_{i,b_1,b_2}\}_{i\in[n],b_1,b_2\in\{0,1\}}, \tilde{t})$.

**Procedure** $\mathsf{randBP}'(\widetilde{\mathsf{BP}})$:

- It picks random and independent non-zero scalars $\{\alpha_{i,b_1,b_2} \in \mathbb{Z}_p\}_{i\in[n],b_1,b_2\in\{0,1\}}$ and computes $C_{i,b_1,b_2} = \alpha_{i,b_1,b_2} \cdot \tilde{B}_{i,b_1,b_2}$. It outputs $(\tilde{s}, \{C_{i,b_1,b_2}\}_{i\in[n],b_1,b_2\in\{0,1\}}, \tilde{t})$.

Before we describe how to simulate the output of $\mathsf{randBP}'$, we will prove a claim about this procedure. Let $M_1, M_2, \ldots, M_n$ be a given set of matrices. Let $(N_1, \ldots, N_n)$ be the output of $\mathsf{randBP}'(M_1, M_2, \ldots, M_n)$. We have that $N_1 = \alpha_1 M_1, N_2 = \alpha_2 M_2, \ldots, N_n = \alpha_n M_n$, where $\alpha_1, \alpha_2, \ldots, \alpha_n$ are non-zero scalars chosen uniformly at random from $\mathbb{Z}_p$. Define $c = (\prod_i N_i)[1,w]$.

**Claim 4.** *If $(\prod_i M_i)[1,w] \neq 0$, then $c$ is distributed uniformly in $\mathbb{Z}_p^*$.*

*Proof.* Since $c = (\prod_i N_i)[1,w] = (\prod_i \alpha_i M_i)[1,w] = (\prod_i \alpha_i)(\prod_i M_i)[1,w]$. Since each $\alpha_i$ is chosen uniformly at random from $\mathbb{Z}_p^*$, $\prod_i \alpha_i$ is distributed uniformly in $\mathbb{Z}_p^*$. Hence, when $(\prod_i M_i)[1,w] \neq 0$, $c$ is distributed uniformly in $\mathbb{Z}_p^*$. □

**Simulator $\mathsf{Sim}'_{\mathsf{BP}}$.** Next, we describe the simulator $\mathsf{Sim}'_{\mathsf{BP}}$ which takes as input $(1^s, \mathsf{BP}(x))$, where $s$ is the size of the formula and $x \in \{0,1\}^\ell$.

$\mathsf{Sim}'_{\mathsf{BP}}(1^s, \mathsf{BP}(x))$:

- If $\mathsf{BP}(x) = 0$, output whatever $\mathsf{Sim}_{\mathsf{BP}}(1^s, 0)$ outputs. Else, pick a $\alpha$ uniformly at random from $\mathbb{Z}_p^*$ and output whatever $\mathsf{Sim}_{\mathsf{BP}}(1^s, \alpha)$ outputs.

Now, we prove the following.

**Theorem 6.** *Consider an oblivious dual-input RMBP $\mathsf{BP} = (\mathsf{inp}_1, \mathsf{inp}_2, \{B_{i,b_1,b_2}\}_{i\in[n],b_1,b_2\in\{0,1\}})$ of length $n$, width $w$ and input of $\ell$ bits. Then there exists a PPT simulator $\mathsf{Sim}'_{\mathsf{BP}}$ such that for every $x \in \{0,1\}^\ell$,*

$$\left\{ \mathsf{randBP}'(\mathsf{randBP}(\mathsf{BP}))\Big|_x \right\} \equiv \left\{ \mathsf{Sim}'_{\mathsf{BP}}(1^s, \mathsf{BP}(x)) \right\}.$$

*Proof.* Let us denote $\mathsf{BP}\Big|_x$ by $(M_1, M_2, \ldots, M_n)$. Observe that

$$\Big\{ \mathsf{randBP}'(\mathsf{randBP}(\mathsf{BP}))\Big|_x \Big\} \equiv \Big\{ \mathsf{randBP}(\mathsf{randBP}'(M_1, M_2, \ldots, M_n)) \Big\}.$$

This holds by just observing that applying $\mathsf{randBP}'(\mathsf{randBP}(\cdot))$ operation on the relaxed matrix branching program and then evaluating the result on an input $x$ is equivalent to first evaluating the relaxed matrix branching program on an input $x$ and then applying the $\mathsf{randBP}'(\mathsf{randBP}(\cdot))$ operation. Now, we need to show that

$$\Big\{ \mathsf{randBP}(\mathsf{randBP}'(M_1, M_2, \ldots, M_n)) \Big\} \equiv \Big\{ \mathsf{Sim}'_{\mathsf{BP}}(1^s, \mathsf{BP}(x)) \Big\}.$$

We will show that for any tuple $V$, the probability of output being $V$ is identical in the real and simulated experiments above. We begin by calculating the probability of $V$ in the real experiment, where probability is taken over the random coins of both $\mathsf{randBP}$ and $\mathsf{randBP}'$. Let $V_2 = M_1, M_2, \ldots, M_n$.

$$\begin{aligned}\Pr[\mathsf{randBP}(\mathsf{randBP}'(V_2)) = V] &= \sum_{V_1} \Pr[\mathsf{randBP}(V_1) = V \wedge \mathsf{randBP}'(V_2) = V_1] \\ &= \sum_{V_1} \Pr[\mathsf{randBP}(V_1) = V] \cdot \Pr[\mathsf{randBP}'(V_2) = V_1]\end{aligned}$$

Now let $V_1 = (N_1, N_2, \ldots, N_n)$ and $\beta_{V_1}$ denote $(\prod_i N_i)[1, w]$. Then by Theorem 5, $\Pr[\mathsf{randBP}(V_1) = V] = \Pr[\mathsf{Sim}_{\mathsf{BP}}(1^s, \beta_{V_1}) = V]$. Substituting in above, we get

$$\begin{aligned}\Pr[\mathsf{randBP}(\mathsf{randBP}'(V_2)) = V] &= \sum_{V_1} \Pr[\mathsf{Sim}_{\mathsf{BP}}(1^s, \beta_{V_1}) = V] \cdot \Pr[\mathsf{randBP}'(V_2) = V_1] \\ &= \sum_{\alpha} \sum_{V_1 s.t. \beta_{V_1} = \alpha} \Pr[\mathsf{Sim}_{\mathsf{BP}}(1^s, \alpha) = V] \cdot \Pr[\mathsf{randBP}'(V_2) = V_1] \\ &= \sum_{\alpha} \Pr[\mathsf{Sim}_{\mathsf{BP}}(1^s, \alpha) = V] \cdot \sum_{V_1 s.t. \beta_{V_1} = \alpha} \Pr[\mathsf{randBP}'(V_2) = V_1]\end{aligned}$$

We have two cases based on whether $\mathsf{BP}(x) = 1$ or $\mathsf{BP}(x) = 0$.

- $\mathsf{BP}(x) = 0$**:** This case is easy to handle. Note that in this case, $\prod_i M_i[1, w] = 0 = \beta_{V_1}$. Hence, in the above expression, $\sum_{V_1 s.t. \beta_{V_1} = \alpha} \Pr[\mathsf{randBP}'(V_2) = V_1] = 1$ for $\beta_{V_1} = 0$ and 0 otherwise. Substituting in the above expression we get,

$$\begin{aligned}\Pr[\mathsf{randBP}(\mathsf{randBP}'(V_2)) = V] &= \Pr[\mathsf{Sim}_{\mathsf{BP}}(1^s, 0) = V] \\ &= \Pr[\mathsf{Sim}'_{\mathsf{BP}}(1^s, \mathsf{BP}(x)) = V]\end{aligned}$$

- $\mathsf{BP}(x) = 1$**:** In this case, $\prod_i M_i[1, w] \neq 0$. By Claim 4, $\sum_{V_1 s.t. \beta_{V_1} = \alpha} \Pr[\mathsf{randBP}'(V_2) = V_1] = \frac{1}{p-1}$. Substituting in above equation we get,

$$\begin{aligned}\Pr[\mathsf{randBP}(\mathsf{randBP}'(V_2)) = V] &= \frac{1}{p-1} \cdot \sum_{\alpha} \Pr[\mathsf{Sim}_{\mathsf{BP}}(1^s, \alpha) = V] \\ &= \Pr[\mathsf{Sim}'_{\mathsf{BP}}(1^s, \mathsf{BP}(x)) = V]\end{aligned}$$

□

# 5 The Ideal Graded Encoding Model

In this section, we describe the ideal graded encoding model. This section has been taken almost verbatim from [BGK+14]. All parties have access to an oracle $\mathcal{M}$, implementing an ideal graded encoding. The oracle $\mathcal{M}$ implements an idealized and simplified version of the graded encoding schemes from [GGH13a]. The parties are provided with encodings of various elements at different levels. They are allowed to perform arithmetic operations of addition/multiplication and testing equality to zero as long as they respect the constraints of the multilinear setting. We start by defining an algebra over the elements.

**Definition 4.** *Given a ring $R$ and a universe set $\mathbb{U}$, an element is a pair $(\alpha, S)$ where $\alpha \in R$ is the* value *of the element and $S \subseteq \mathbb{U}$ is the* index *of the element. Given an element $e$ we denote by $\alpha(e)$ the value of the element, and we denote by $S(e)$ the index of the element. We also define the following binary operations over elements:*

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

Next, we describe the oracle $\mathcal{M}$. $\mathcal{M}$ is a stateful oracle mapping elements to "generic" representations called *handles*. Given handles to elements, $\mathcal{M}$ allows the user to perform operations on the elements. $\mathcal{M}$ will implement the following interfaces:

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

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

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

# 6 Straddling Set System

In this section, we describe a straddling set system which is same as the one considered in [BGK+14]. Then we will prove two combinatorial properties of this set system, which will be very useful in proving the VBB security of our scheme.

**Definition 5.** *A straddling set system $\mathbb{S}_n = \{S_{i,b} : i \in [n], b \in \{0,1\}\}$ with $n$ entries over the universe $\mathbb{U} = \{1, 2, \ldots, 2n-1\}$ is as follows:*

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

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

**Claim 5** (Two unique covers of universe)**.** *The only exact covers of* $\mathbb{U}$ *are* $\{S_{i,0}\}_{i\in[n]}$ *and* $\{S_{i,1}\}_{i\in[n]}$.

*Proof.* Since any exact cover of $\mathbb{U}$ needs to pick a set with element 1, it either contains the set $S_{1,0}$ or $S_{1,1}$. Let $\mathcal{C}$ be a cover of $\mathbb{U}$ containing $S_{1,0}$. Then, we prove that $S_{i,0} \in \mathcal{C}, \forall i \in [n]$. We will prove this via induction on $i$. It is trivially true for $i = 1$. Let us assume that the statement is true for $i$, and prove the statement for $i + 1$. There are only two sets, namely $S_{i+1,0}$ and $S_{i,1}$ which contain the element $2i \in \mathbb{U}$. Since, by induction hypothesis, $S_{i,0} \in \mathcal{C}$ and $S_{i,0} \cap S_{i,1} \neq \emptyset$, $S_{i+1,0} \in \mathcal{C}$ in order to cover all the elements in $\mathbb{U}$. This shows that there is a unique cover of $\mathbb{U}$ containing $S_{1,0}$.

Similarly, we can show that there is a unique cover of $\mathbb{U}$ containing the set $S_{1,1}$ which is $\{S_{i,1}\}_{i\in[n]}$. As mentioned before, any exact cover of $\mathbb{U}$ contains either $S_{1,0}$ or $S_{1,1}$ in order to cover the element $1 \in \mathbb{U}$. This proves the claim. □

**Claim 6** (Collision at universe)**.** *Let* $\mathcal{C}$ *and* $\mathcal{D}$ *be non-empty collections of sets such that* $\mathcal{C} \subseteq \{S_{i,0}\}_{i\in[n]}$, $\mathcal{D} \subseteq \{S_{i,1}\}_{i\in[n]}$, *and* $\bigcup_{S\in\mathcal{C}} S = \bigcup_{S\in\mathcal{D}} S$, *then following must hold:*

$$\mathcal{C} = \{S_{i,0}\}_{i\in[n]} \ , \ \mathcal{D} = \{S_{i,1}\}_{i\in[n]}.$$

*Proof.* We will prove this claim by contradiction. Let us assume that $\mathcal{C} \subset \{S_{i,0}\}_{i\in[n]}$. Then there exists a maximal sub-interval $[i, j] \subset [n]$ such that $S_{k,0} \in \mathcal{C}$ for all $i \leq k \leq j$ but *either* (1) $i > 1$ and $S_{i-1,0} \notin \mathcal{C}$ *or* (2) $j < n$ and $S_{j+1,0} \notin \mathcal{C}$.

(1) Since $(2i - 2) \in S_{i,0} \in \mathcal{C}$ and $\bigcup_{S\in\mathcal{C}} S = \bigcup_{S\in\mathcal{D}} S$, it should be the case that $S_{i-1,1} \in \mathcal{D}$. Now by a similar argument, since $(2i - 3) \in S_{i-1,1} \in \mathcal{D}$ and $\bigcup_{S\in\mathcal{C}} S = \bigcup_{S\in\mathcal{D}} S$, it should be the case that $S_{i-1,0} \in \mathcal{C}$. This contradicts the assumption that $i > 1$ and $S_{i-1,0} \notin \mathcal{C}$.

(2) Since $(2j - 1) \in S_{j,0} \in \mathcal{C}$ and $\bigcup_{S\in\mathcal{C}} S = \bigcup_{S\in\mathcal{D}} S$, it should be the case that $S_{j,1} \in \mathcal{D}$. Now by a similar argument, since $(2j) \in S_{j,1} \in \mathcal{D}$ and $\bigcup_{S\in\mathcal{C}} S = \bigcup_{S\in\mathcal{D}} S$, it should be the case that $S_{j+1,0} \in \mathcal{C}$. This contradicts the assumption that $j < n$ and $S_{j+1,0} \notin \mathcal{C}$.

Since $\mathcal{C} = \{S_{i,0}\}_{[}n]$, it has to be the case that $\mathcal{D} = \{S_{i,1}\}_{[}n]$. □

# 7 Obfuscation in the Ideal Graded Encoding Model

In this section, we describe our VBB obfuscator $\mathcal{O}$ for polynomial sized formulae in the ideal graded encoding model.

**Input.** The input to our obfuscator $\mathcal{O}$ is a polynomial size formula $F$ which takes $\ell$ bit inputs. $\mathcal{O}$ first converts it to a layered branching program as described in Section 3.1. Next, it converts this to a relaxed matrix branching program as described in Section 3.2. Finally, it converts it to a dual-input oblivious relaxed matrix branching program $\mathsf{BP}$ of length $n$ and width $w$, i.e.

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

such that $\mathsf{inp}_1$ and $\mathsf{inp}_2$ are evaluation functions mapping $[n] \to [\ell]$, each $B_{i,b_1,b_2} \in \{0, 1\}^{w\times w}$ is a full rank matrix and the following conditions are satisfied

1. For each $i \in [n]$, $\mathsf{inp}_1(i) \neq \mathsf{inp}_2(i)$.
2. Each pair of input bits is paired at some index of $\mathsf{BP}$. That is, for each $(j, k) \in [\ell] \times [\ell]$ there exists an index $i$ such that either $\mathsf{inp}_1(i) = j$ and $\mathsf{inp}_2(i) = k$ *or* $\mathsf{inp}_1(i) = k$ and $\mathsf{inp}_2(i) = j$.

3. Each input bit is used for exactly $\ell'$ indices of $\mathsf{BP}$. More precisely, for each $j \in [\ell]$, define $\mathsf{ind}(j) = \{i \in [n] \ : \ \mathsf{inp}_1(i) = j\} \cup \{i \in [n] \ : \ \mathsf{inp}_2(i) = j\}$. Then this condition implies that for all $j \in [\ell]$, $|\mathsf{ind}(j)| = \ell'$.

Note that the last condition is not necessary for our proof to go through. Yet we assume it for the ease of notation in our proof.

Also observe from Section 3 that evaluation functions $\mathsf{inp}_1, \mathsf{inp}_2$, length $n$ and width $w$ of the RMBP depend only on the size $s$ of the formula $F$.

**Randomizing the relaxed matrix branching program $\mathsf{BP}$.** The obfuscator $\mathcal{O}$ randomizes the branching program in two steps using procedures $\mathsf{randBP}$ and $\mathsf{randBP}'$ described in Section 4. It begins by sampling a prime $p$ of $O(n)$ bits.

1. It invokes the procedure $\mathsf{randBP}$ on the relaxed matrix branching program $\mathsf{BP}$ obtained above to get $\left(\tilde{s}, \left\{\tilde{B}_{i,b_1,b_2}\right\}_{i\in[n],b_1,b_2\in\{0,1\}}, \tilde{t}\right)$. Recall that $\tilde{s}, \tilde{t} \in \mathbb{Z}_p^w$ and $\tilde{B}_{i,b_1,b_2} \in \mathbb{Z}_p^{w\times w}$ for all $i \in [n], b_1, b_2 \in \{0,1\}$.
2. It picks random and independent non-zero scalars $\{\alpha_{i,b_1,b_2} \in \mathbb{Z}_p\}_{i\in[n],b_1,b_2\in\{0,1\}}$. Define $C_{i,b_1,b_2} = \alpha_{i,b_1,b_2} \cdot \tilde{B}_{i,b_1,b_2}$.

The output of the randomization phase is $(\tilde{s}, \{C_{i,b_1,b_2}\}_{i\in[n],b_1,b_2\in\{0,1\}}, \tilde{t})$.[6]

The final obfuscation of the formula $F$ will consists of ideal encodings of these elements with respect to a carefully chosen set system. Next, we describe how these sets are chosen.

**Initialization.** Consider a universe set $\mathbb{U}$. Let $\mathbb{U}_s, \mathbb{U}_t, \mathbb{U}_1, \mathbb{U}_2, \ldots, \mathbb{U}_\ell$ be partitions of $\mathbb{U}$ such that for all $j \in [\ell]$, $|\mathbb{U}_j| = (2\ell' - 1)$. That is, $\mathbb{U}_s, \mathbb{U}_t, \mathbb{U}_1, \mathbb{U}_2, \ldots, \mathbb{U}_\ell$ are disjoint sets and $\mathbb{U} = \mathbb{U}_s \cup \mathbb{U}_t \cup \bigcup_{j=1}^{\ell} \mathbb{U}_j$.

Now let $\mathbb{S}^j$ be the straddling set system (defined in Section 6) over the elements in $\mathbb{U}_j$. Note that $\mathbb{S}^j$ will have $\ell'$ entries which is same as $|\mathsf{ind}j|$ for each $j \in [\ell]$. We now associate the entries in the straddling set system $\mathbb{S}^j$ with the indices of $\mathsf{BP}$ which depend on $x_j$, i.e. the set $\mathsf{ind}(j)$. More precisely, let

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

Next, we associate a set to each element output by the randomization step. Recall that in a dual-input relaxed matrix branching program, each step depends on two fixed bits in the input defined by the evaluation functions $\mathsf{inp}_1$ and $\mathsf{inp}_2$. For each step $i \in [n]$, $b_1, b_2 \in \{0,1\}$, we define the set $S(i, b_1, b_2)$ using the straddling sets for input bits $\mathsf{inp}_1(i)$ and $\mathsf{inp}_2(i)$ as follows:

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

We will use the set $S(i, b_1, b_2)$ to encode the elements of $C_{i,b_1,b_2}$. We will use the sets $\mathbb{U}_s$ and $\mathbb{U}_t$ to encode the elements in $\tilde{s}$ and $\tilde{t}$ respectively. More formally, $\mathcal{O}$ does the following:

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

$$\left\{(\tilde{s}[k], \mathbb{U}_s), \ (\tilde{t}[k], \mathbb{U}_t)\right\}_{k\in[w]}$$

$$\{(C_{i,b_1,b_2}[j,k], S(i, b_1, b_2)\}_{i\in[n],b_1,b_2\in\{0,1\},j,k\in[w]}$$

[6] This step is identical to the procedure $\mathsf{randBP}'$ described in Section 4.

$\mathcal{O}$ receives a list of handles for these elements from $\mathcal{M}$. Let $[\beta]_S$ denote the handle to $(\beta, S)$. For a matrix $M$, let $[M]_S$ denote a matrix of handles such that $[M]_S[j,k]$ is a handle for $(M[j,k], S)$. Thus, $\mathcal{O}$ receives the following handles.

$$[\tilde{s}]_{\mathbb{U}_s}, \quad [\tilde{t}]_{\mathbb{U}_t}, \quad \left\{[C_{i,b_1,b_2}]_{S(i,b_1,b_2)}\right\}_{i\in[n],b_1,b_2\in\{0,1\}}$$

**Output.** $\mathcal{O}$ outputs these handles as the obfuscation for the formula $F$. Given access to the oracle $\mathcal{M}$, $\mathcal{O}(F)$ can add and multiple handles and test for zero at the universe set.

Now we state our main theorems as follows:

**Theorem 7.** *There exists a obfuscator $\mathcal{O}$ which is a virtual black box obfuscator in the idealized model for the class of all polynomial sized boolean formulae according to Definition 3.*

**Theorem 8.** *There exists a obfuscator $\mathcal{O}$ which is a virtual black box obfuscator in the idealized model for the class of all polynomial sized relaxed matrix branching programs (Section 2.3) according to Definition 3.*

**Evaluation of $\mathcal{O}(F)$ on input $x$.** Recall that two handles corresponding to the same set $S$ can be added. If $[\beta]_S$ and $[\gamma]_S$ are two handles, we denote the handle for $(\beta+\gamma, S)$ obtained from $\mathcal{M}$ on addition query by $[\beta]_S + [\gamma]_S$. Similarly, two handles corresponding to $S_1$ and $S_2$ can be multiplied if $S_1 \cap S_2 = \emptyset$. We denote the handle for $(\beta\cdot\gamma, S_1 \cup S_2)$ obtained from $\mathcal{M}$ on valid multiplication query on $[\beta]_{S_1}$ and $[\gamma]_{S_2}$ by $[\beta]_{S_1}\cdot[\gamma]_{S_2}$. Similarly, we denote the handle for $(M_1\cdot M_2, S_1\cup S_2)$ by $[M_1]_{S_1}\cdot[M_2]_{S_2}$.

Given $x\in\{0,1\}^\ell$, to compute $F(x)$, $\mathcal{O}(F)$ computes the handle for the following expression:

$$h = [\tilde{s}]_{\mathbb{U}_s}\cdot\prod_{i=1}^{n}\left[C_{i,x_{\mathsf{inp}_1(i)},x_{\mathsf{inp}_2(i)}}\right]_{S(i,x_{\mathsf{inp}_1(i)},x_{\mathsf{inp}_2(i)})}\cdot[\tilde{t}]_{\mathbb{U}_t}$$

Next, $\mathcal{O}(F)$ uses the oracle $\mathcal{M}$ to do a zero-test on $h$. If the zero-test returns a 1, then $\mathcal{O}(F)$ outputs 0 else it outputs 1.

In the next section, define $b_1^i = x_{\mathsf{inp}_1(i)}$ and $b_2^i = x_{\mathsf{inp}_2(i)}$.

**Correctness of Evaluation.** We will first assume that none of the calls to $\mathcal{M}$ fail and show that $\mathcal{O}(F)$ on $x$ outputs 1 iff $F(x)=1$. From the description of the evaluation above, $\mathcal{O}(F)$ outputs 0 on $x$ if and only if

$$\begin{aligned}
0 &= \tilde{s}\cdot\prod_{i=1}^{n} C_{i,b_1^i,b_2^i}\cdot\tilde{t}\\
&= \tilde{s}\cdot\prod_{i=1}^{n}\alpha_{i,b_1^i,b_2^i}\cdot\tilde{B}_{i,b_1^i,b_2^i}\cdot\tilde{t}\\
&= \left((e_1\cdot R_0^{-1})\cdot\prod_{i=1}^{n} R_{(i-1)}\cdot B_{i,b_1^i,b_2^i}\cdot R_i^{-1}\cdot(R_n\cdot e_w)\right)\cdot\prod_{i=1}^{n}\alpha_{i,b_1^i,b_2^i}\\
&= \left(e_1\cdot\prod_{i=1}^{n} B_{i,b_1^i,b_2^i}\cdot e_w\right)\cdot\prod_{i=1}^{n}\alpha_{i,b_1^i,b_2^i}\\
&= P_x[1,w]\cdot\prod_{i=1}^{n}\alpha_{i,b_1^i,b_2^i}
\end{aligned}$$

From Lemmas 1 and 2, we have that $P_x[1, w] = 0$ if and only if $F(x) = 0$. Hence, using the fact that each $\alpha_{i,b_1^i,b_2^i}$ is a non-zero scalar, the correctness of evaluation holds.

Now we are left to show that no call to oracle $\mathcal{M}$ fails. To show that none of the oracle calls made while computing the handle $h$ fail, we need to show that the following sets are disjoint for any $x \in \{0,1\}^\ell$: $\mathbb{U}_s, \mathbb{U}_t, \{S(i, b_1^i, b_2^i)\}_{i\in[n]}$.

Note that from the construction of the straddling set system, it holds that for any $\mathbb{S}^j$, and any two $i, i' \in \mathsf{ind}(j)$ and any $b \in \{0,1\}$, $S^j_{i,b} \cap S^j_{i',b} = \emptyset$. Now disjointness of $\mathbb{U}_s, \mathbb{U}_t, \{S(i, b_1^i, b_2^i)\}_{i\in[n]}$ holds by using how each $S(i, b_1^i, b_2^i)$ was constructed and the fact that $\mathbb{U}_s, \mathbb{U}_t, \mathbb{U}_1, \ldots, \mathbb{U}_\ell$ are disjoint.

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

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

which follows from the following equalities:

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

where the first equality follows from definition of $S(i, b_1, b_2)$, second and third equality follow from how the straddling set system $\mathbb{S}^j$ corresponding to $x_j$ is constructed.

# 8 Proof of Virtual Black Box Obfuscation in the Idealised Graded Encoding Model

In this section, we prove that the obfuscator $\mathcal{O}$ described in Section 7 is a good VBB obfuscator for polynomial sized formulas in the ideal graded encoding model.

Let $\mathcal{F} = \{\mathcal{F}_\ell\}_{\ell\in\mathbb{N}}$ be a formula class such that every formula in $\mathcal{F}_\ell$ is of size $O(\ell)$. We assume WLOG that all formulas in $\mathcal{F}_\ell$ are of the same size (otherwise the formula can be padded). It follows from Theorem 3 that for any formula $F$ there exists a RMBP represented in the form of $O(|F|)$ matrices each of width $O(|F|)$. Hence, there exists linear functions $n(\cdot)$ and $w(\cdot)$ such that $\mathcal{O}$ in Section 7 outputs a dual-input oblivious RMBP of size $n(|F|)$ and width $w(|F|)$ computing on $\ell(|F|)$ inputs. Hence, $\mathcal{O}$ satisfies the polynomial slowdown requirement. We also showed that $\mathcal{O}$ satisfies the functionality requirement and always computes the correct output (see Section 7). We are now left to show that $\mathcal{O}$ satisfies the virtual black box property.

**The Simulator $\mathsf{Sim}$.** Here we construct a simulator $\mathsf{Sim}$ that takes as input $1^{|F|}$ and description of the adversary $\mathcal{A}$, and is given oracle access to the formula $F$. This simulator is required to simulate the view of the adversary.

The simulator begins by emulating the obfuscator $\mathcal{O}$ on $F$. First, the simulator needs to compute the RMBP $\mathsf{BP}_F$ and the matrices $B_{i,b_1,b_2}$ corresponding to the branching program. Note that the simulator is only given oracle access to the formula $F$ and has no way to compute these matrices. Thus, $\mathsf{Sim}$ initializes the oracle $\mathcal{M}$ with formal variables. Also note that the simulator can compute the evaluation functions $\mathsf{inp}_1$ and $\mathsf{inp}_2$ and also the system used for encodings since the RMBPs are oblivious. This would be important when $\mathsf{Sim}$ simulates the oracle queries of $\mathcal{A}$.

More formally, we extend the definition of an element to allow for values that are formal variables and also expressions over formal variables, instead of just being ring elements. When we

perform an operation $\circ$ on two elements $e_1$ and $e_2$, that contain formal variables, the resultant element $e_1 \circ e_2$ is a corresponding arithmetic expression over formal variables. This way we represent formal expressions as arithmetic circuits. We denote by $\alpha(e)$ the arithmetic expression over formal variables for element $e$. An element is called *basic* element if the corresponding arithmetic circuit has no gates, i.e. either it is a constant or a formal variable. We say that $e'$ is a *sub-element* of $e$ if the circuit corresponding to $e'$ is a sub-circuit of the circuit for $e$.

Next, $\mathsf{Sim}$ will emulate the oracle $\mathcal{M}$ that $\mathcal{O}$ accesses as follows: $\mathsf{Sim}$ will maintain a table of handles and corresponding level of encodings that have been initialized so far. As mentioned before, $\mathsf{Sim}$ will initialize the oracle $\mathcal{M}$ with formal variables. Note that $\mathsf{Sim}$ can emulate all the interfaces of $\mathcal{M}$ apart from the zero-testing. Note that $\mathcal{O}$ does not make any zero-test queries. Hence, the simulation of the obfuscator $\mathcal{O}$ is perfect.

When $\mathsf{Sim}$ completes the emulation of $\mathcal{O}$ it obtains a simulated obfuscation $\tilde{\mathcal{O}}(F)$. Now $\mathsf{Sim}$ has to simulate the view of the adversary on input $\tilde{\mathcal{O}}(F)$. Our $\mathsf{Sim}$ will use the same handles table for emulating the oracle calls of both $\mathcal{O}$ and $\mathcal{A}$. Hence, $\mathsf{Sim}$ can perfectly emulate all the oracle calls made by $\mathcal{A}$ apart from zero-testing. The problem with answering zero-test queries is that $\mathsf{Sim}$ cannot zero-test the expressions involving formal variables. Zero-testing is the main challenge for simulation, which we describe in the next section. Since the distribution of handles generated during the simulation and during the real execution are identical, and since the obfuscation consists only of handles (as opposed to elements), we have that the simulation of the obfuscation $\tilde{\mathcal{O}}$ and the simulation of $\mathcal{M}$'s answers to all the queries, except for zero-test queries, is perfect.

**Simulating zero testing queries.** In this part we describe how our simulator handles the zero-test queries made by $\mathcal{A}$. This part is the non-trivial part of the analysis for the following reason. The handle being zero-tested is an arithmetic circuit whose value depends on the formal variables which are unknown to the simulator. The real value for these formal variables would depend on the formula $F$. At a very high level, we show that these values can be simulated given oracle access to $F$.

There are two steps to zero-testing an element. Note that the adversary may have combined the handles provided in very convoluted manner. More precisely, $\mathcal{A}$ may have computed sub-expressions involving multiple inputs and hence, the value of the element being zero-tested may depend on formal variables which correspond to using multiple inputs. Hence, the first step is to decompose this elements into "simpler" elements that we call *single-input elements*. As the name suggests, any single input element's circuit consists of formal variables corresponding to a distinct input $x \in \{0,1\}^\ell$. Namely, it only depends on formal variables in matrices $C_{i,b_1,b_2}$ such that $b_1 = x_{\mathsf{inp}_1(i)}$ and $b_2 = x_{\mathsf{inp}_2(i)}$. In the first step we show that any element $e$, such that $S(e) = \mathbb{U}$ which is zero-tested can be decomposed into polynomial number of single input elements.

In the second step, $\mathsf{Sim}$ simulates the value of each of the single input elements obtained via decomposition independently. More formally, we use Theorem 6 to show that value of each single-input element can be simulated perfectly. But we run into the following problem. We cannot simulate the value of all the single input elements together as these have correlated randomness of the obfuscator. Instead we show that it suffices to zero-test each single-input element individually. For this we use the fact that each of the matrix $\tilde{B}_{i,b_1,b_2}$ was multiplied by $\alpha_{i,b_1,b_2}$. Using this we prove that value of each single input element depends on product of different $\alpha$'s which is determined by the input being used. Now, we use the fact that the probability that $\mathcal{A}$ creates an element such that non-zero value of two single input elements cancel each other is negligible. Therefore, it holds that element is zero iff each of the single input elements are zero independently.

## 8.1 Decomposition to Single-Input Elements

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

For every element $e$, we will assign an *input-profile* $\mathsf{Prof}(e) \in \{0,1,*\}^\ell \cup \{\perp\}$. Intuitevely, if $e$ is a sub-expression in the evaluation of the obfuscated program on some input $x \in \{0,1\}^\ell$, then $\mathsf{Prof}(e)$ is used to represent the partial information about $x$ which can be learnt from formal variables which occur in $e$. For example, we say that $\mathsf{Prof}(e)_j$ is *consistent* with the bit $b$ if there exists a basic sub-element $e'$ of $e$ such that $S(e') = S(i, b_1, b_2)$ such that $\mathsf{inp}_1(i) = j$ and $b_1 = b$ or $\mathsf{inp}_2(i) = j$ and $b_2 = b$. Next, for every $j \in [\ell]$ we set $\mathsf{Prof}(e)_j = b$ iff $\mathsf{Prof}(e)_j$ is consistent with $b$ and is not consistent with $(1-b)$. If $\mathsf{Prof}(e)_j$ is neither consistent with $b$ nor $(1-b)$, we set $\mathsf{Prof}(e)_j = *$. Finally, we set $\mathsf{Prof}(e) = \perp$ iff there exists a $j \in [\ell]$ such that $\mathsf{Prof}(e)$ is consistent with both $b$ and $(1-b)$. We call $e$ a *single-input* element iff $\mathsf{Prof}(e) \neq \perp$. Finally, if $\mathsf{Prof}(e) \in \{0,1\}^\ell$, we say that input-profile of $e$ is *complete*. Otherwise, we say that input-profile of $e$ is *partial*.

We also define the partial symmetric operation $\odot : \{0,1,*,\perp\} \times \{0,1,*,\perp\} \to \{0,1,\perp\}$ as follows: $b \odot * = b$ for $b \in \{0,1,*,\perp\}$, $b \odot b = b$, and $b \odot (1-b) = \perp$ for $b \in \{0,1\}$, and $\perp \odot \perp = \perp$. If $\odot$ is applied to two vectors, it is performed separately for each position.

Next we describe an algorithm $D$ used by $\mathsf{Sim}$ to decompose elements into single-input elements. Parts of this description have been taken verbatim from [BGK+14]. Given an element $e$, $D$ outputs a set of single-input elements with distinct input-profiles such that $e = \sum_{s \in D(e)} s$, where the equality between the elements means that their values compute the same function (it does not mean that the arithmetic circuits that represent these values are identical). Note that the above requirement implies that for every $s \in D(e)$, $S(s) = S(e)$. Moreover, for each $s \in D(e)$, $D$ also computes the input-profile of $s$ recursively.

The decomposition algorithm $D$ outputs a set of elements and their associated input profile and is defined recursively, as follows:

- Element $e$ is basic: $D$ outputs the singleton set $\{e\}$. Let $S(e) = S(i, b_1, b_2)$. Then $\mathsf{Prof}(e)$ is as follows: $\mathsf{Prof}(e)_{\mathsf{inp}_1(i)} = b_1$, $\mathsf{Prof}(e)_{\mathsf{inp}_2(i)} = b_2$, and $\mathsf{Prof}(e)_j = *$ for all $j \in [\ell], j \neq \mathsf{inp}_1(i), j \neq \mathsf{inp}_2(i)$.
- Element $e$ is of the form $e_1 + e_2$: $D$ computes recursively $L_1 = D(e_1), L_2 = D(e_2)$ and outputs $L = L_1 \cup L_2$. If there exist elements $s_1, s_2 \in L$ with the same input-profile, $D$ replaces the two elements with a single element $s = s_1 + s_2$ and $\mathsf{Prof}(s) = \mathsf{Prof}(s_1)$. It repeats this process until all the input-profiles in $L$ are distinct and outputs $L$.
- Element $e$ is of the form $e_1 \cdot e_2$: $D$ computes recursively $L_1 = D(e_1), L_2 = D(e_2)$. For every $s_1 \in L_1$ and $s_2 \in L_2$, $D$ adds the expression $s_1 \cdot s_2$ to the output set $L$ and sets $\mathsf{Prof}(s) = \mathsf{Prof}(s_1) \odot \mathsf{Prof}(s_2)$. $D$ then eliminates repeating input-profiles from $L$ as described above, and outputs $L$.

**Remark 2.** *Note that if $s = s_1 \cdot s_2$ such that $\mathsf{Prof}(s_1)_j = 0$ and $\mathsf{Prof}(s_2)_j = 1$, then $\mathsf{Prof}(s)_j = \perp$. Hence, multiplication gates can lead to an element with* invalid *input-profile. This observation will be used often in the later proofs.*

The fact that in the above decomposition algorithm indeed $e = \sum_{s \in D(e)} s$, and that the input profiles are distinct follows from a straightforward induction. Now, we prove a set of claims and conclude that $D(e)$ runs in polynomial time (see Claim 9). We begin by proving a claim about the relation between the level of encoding of $e$ and a sub-element $e'$ of $e$.

**Claim 7.** *If $e'$ is a sub-element of $e$, then there exists a collection of disjoint sets $\mathcal{C}$ from our set systems $\{\mathbb{S}^j\}_{j \in [\ell]}$, $\mathbb{U}_s$ and $\mathbb{U}_t$ such that the sets in $\mathcal{C}$ are disjoint with $S(e')$ and $S(e) = S(e') \cup \bigcup_{S \in \mathcal{C}} S$.*

The above claim says that if $e'$ is a sub-element of $e$, the set corresponding to the encoding of $e$ can be seen as being derived from the set used for encoding of $e'$. Intuitively, this is true because in obtaining $e$ from $e'$, the set of encoding never shrinks. It remains same with each addition and increases as union of two disjoint sets with each multiplication. Thus, there would exist a collection of sets such that $S(e)$ can be written as the union of this collection of disjoint sets along with the set of $e'$. In other words, there exists a cover for $S(e)$ which involves the set $S(e')$ and some other disjoint sets from our set system.

*Proof.* (of Claim 7) We will prove this claim by induction on the size of $e$. If $e = 1$, i.e. $e$ is a basic element, then the claim trivially holds. If $e = e_1 + e_2$, then either (1) $e' = e$ or (2) $e'$ is a sub-element of either $e_1$ or $e_2$. In the first case, the claim is trivially true. In the second case, let wlog $e'$ be sub-element of $e_1$. Then by induction hypothesis, there exists a collection of disjoint sets $\mathcal{C}$ from our set systems such that the sets in $\mathcal{C}$ are disjoint with $S(e')$ and $S(e_1) = S(e') \cup \bigcup_{S \in \mathcal{C}} S$. The claim follows by noting that $S(e) = S(e_1)$.

Finally, if $e = e_1 \cdot e_2$, either (1) $e' = e$ or (2) $e'$ is a sub-element of either $e_1$ or $e_2$. In the first case, the claim is trivially true. In the second case, let wlog $e'$ be sub-element of $e_1$. Then by induction hypothesis, there exists a collection of disjoint sets $\mathcal{C}_1$ from our set systems such that the sets in $\mathcal{C}_1$ are disjoint with $S(e')$ and $S(e_1) = S(e') \cup \bigcup_{S \in \mathcal{C}_1} S$. Now, for $e_2$ either (1) $e_2$ is a basic element or (2) there exists a basic sub-element $e''$ of $e_2$. In the first case, $\mathcal{C} = \mathcal{C}_1 \cup \{S(e_2)\}$ since for valid multiplication $S(e_1) \cap S(e_2) = \emptyset$. In the second case, we apply the induction hypothesis on $e_2, e''$ and get a collection of sets $\mathcal{C}_2$ and $\mathcal{C} = \mathcal{C}_1 \cup (S(e'') \cup \mathcal{C}_2)$. Note that $S(e'')$ is a union of two disjoint sets from our set system. □

Next, we prove that for elements which can be zero-tested, i.e. elements at the highest level of encoding, all the elements output by the procedure $D$ are single input elements. In this direction, we first observe that adding two elements does not create new input-profiles. That is, only way to create new profiles is to multiply two elements. As noted in Remark 2, multiplication of two elements can lead to invalid profiles. Here we use the observation that if $e = e_1 \cdot e_2$ has invalid input profile then computations involving $e$ cannot lead to an element at the universe set and cannot be zero-tested. Here we crucially use the properties of straddling sets and Claim 7. More formally,

**Claim 8.** *If $\mathbb{U} = S(e)$ then all the elements in $D(e)$ are single-input elements. Namely, for every $s \in D(e)$ we have that $\mathsf{Prof}(s) \neq \perp$.*

*Proof.* We will prove this claim by contradiction. Let us assume that the claim is false. Then there exists a sub-element $e^{\mathsf{bad}}$ of $e$ such that $D(e^{\mathsf{bad}})$ contains an invalid input-profile but decomposition of all sub-elements of $e^{\mathsf{bad}}$ have valid input-profiles. We now do a case analysis on the structure of $e^{\mathsf{bad}}$.

$e^{\mathsf{bad}}$ cannot be a basic sub-element since input-profile of all basic sub-elements is valid. Also, $e^{\mathsf{bad}}$ cannot be of the form $e_1 + e_2$ because input-profiles in $D(e^{\mathsf{bad}})$ is a union of input-profiles in $D(e_1)$ and $D(e_2)$. Hence, $e^{\mathsf{bad}}$ is of the form $e_1 \cdot e_2$.

The only way $D(e^{\mathsf{bad}})$ contains an invalid input-profile when all input profiles in $D(e_1)$ and $D(e_2)$ are valid is the following: There exists a $s_1 \in D(e_1)$ and $s_2 \in D(e_2)$ such that $\mathsf{Prof}(s_1) \neq \perp$ and $\mathsf{Prof}(s_2) \neq \perp$ but $\mathsf{Prof}(s_1 \cdot s_2) = \perp$. Then, wlog there exists $j \in [\ell]$ such that $\mathsf{Prof}(s_1) = 0$ and $\mathsf{Prof}(s_2) = 1$. From the description of input profiles, there exists a basic sub-element $\hat{e_1}$ of $s_1$ such that $S(\hat{e_1}) \cap \mathbb{U}_j = S^j_{k,0} \in \mathbb{S}^j$ for some $k \in \mathsf{ind}(j)$. Similarly, there exists a basic sub-element $\hat{e_2}$ of $s_2$ such that $S(\hat{e_2}) \cap \mathbb{U}_j = S^j_{k',1} \in \mathbb{S}^j$ for some $k \in \mathsf{ind}(j)$.

Intuitively, using Claim 5, we show that there is no way of combining $\hat{e_1}$ and $\hat{e_2}$ to form a valid element $e$ such that $S(e) \supseteq \mathbb{U}_j$. For this, we critically use the properties of the straddling set system and the fact that the set used for encoding only grows as union of two disjoint sets (as we do more multiplications). Hence, to obtain $e$ using $\hat{e_1}$ and $\hat{e_2}$, we need to find a collection of

disjoint sets whose union along with $S(\hat{e_1})$ and $S(\hat{e_2})$ gives $\mathbb{U}$. This is not possible by properties of straddling sets. More formally, we have the following:

Since, $\hat{e_1}$ is a basic sub-element of $s_1$, by Claim 7, there exists a collection $\mathcal{C}_1$ such that $S(s_1) = S(\hat{e_1}) \cup \bigcup_{S \in \mathcal{C}_1} S$. Similarly, there exists a collection $\mathcal{C}_2$ such that $S(s_2) = S(\hat{e_2}) \cup \bigcup_{S \in \mathcal{C}_2} S$. Since $(s_1 \cdot s_2)$ is a valid multiplication, $\left(S(\hat{e_1}) \cup \bigcup_{S \in \mathcal{C}_1} S\right) \bigcup \left(S(\hat{e_2}) \cup \bigcup_{S \in \mathcal{C}_2} S\right) = S(s_1 \cdot s_2) = S(e_1 \cdot e_2) = S(e^{\mathsf{bad}})$.

Again, since $e^{\mathsf{bad}}$ is a sub-element of $e$, using Claim 7, there exists a collection $\mathcal{C}$ such that $S(e^{\mathsf{bad}})$ and $\mathcal{C}$ form a cover for $S(e)$. This implies that there is an exact cover of $\mathbb{U}$ using both $S^j_{k,0}$ and $S^j_{k',1}$ for some $k, k' \in \mathsf{ind}(j), j \in [\ell]$. This is a contradiction to Claim 5 for straddling set system $\mathbb{S}^j$ for $\mathbb{U}_j$. □

Finally, we prove the main claim of this section that $D$ runs in polynomial time. First observe that only multiplication can create new input profiles. We show that if $e$ is an element of the form $e_1 \cdot e_2$ and $D(e)$ contains a new input-profile then $e$ must itself be a single-input element (that is, $D(e)$ will be the singleton set $\{e\}$). This means that the number of elements in the decomposition of $e$ is bounded by the number of sub-elements of $e$, and therefore is polynomial. To prove the above we first observe that if $D(e)$ is not a singleton, then either $D(e_1)$ or $D(e_2)$ are also not singletons. Then we show that if $D(e_1)$ contains more than one input-profile then all input-profiles in $D(e_1)$ must be complete. Here again we use the structure of the straddling set system and therefore the multiplication $e_1 \cdot e_2$ cannot contain any new profiles.

**Claim 9.** *$D(e)$ runs in polynomial time, i.e. number of elements in $D(e)$ is polynomial.*

*Proof.* Observe that the running time of $D$ on $e$ is polynomial in the number of the single-input elements in $D(e)$. Hence, to show that $D$ runs in polynomial time, we will show that the size of the set $D(e)$ is bounded by the number of sub-elements of $e$. More precisely, for each $s \in D(e)$, we show a single-input sub-element $e'$ of $e$ such that $\mathsf{Prof}(e') = \mathsf{Prof}(s)$. Since $D(e)$ has single input elements with distinct profiles, we get that $|D(e)|$ is polynomial since $e$ has a polynomial number of sub-elements.

For each $s \in D(e)$, let $e'$ be the first sub-element of $e$ such that $D(e')$ contains a single input element with input-profile $\mathsf{Prof}(s)$ and decomposition of no sub-element of $e'$ contains a single-input element with input-profile $\mathsf{Prof}(s)$. Then we claim that $e'$ is a single input element, i.e. $D(e') = \{e'\}$. We have the following cases.

$e'$ is a basic sub-element of $e$, then by definition, $D(e') = \{e'\}$. Next, if $e' = e_1 + e_2$, then all the input-profiles in $D(e')$ are either in $e_1$ or $e_2$. That is, $e'$ cannot be the first sub-element of $e$ which contains the input profile $\mathsf{Prof}(s)$. Finally, let $e' = e_1 \cdot e_2$. We need to show that $D(e') = \{e'\}$. Suppose not, that is $|D(e')| > 1$. In this case, we will show that $D(e')$ cannot contain any new input profiles. Let $s' \in D(e')$ such that $\mathsf{Prof}(s) = \mathsf{Prof}(s')$.

By the definition of $D$, either $|D(e_1)| > 1$ or $D(e_2) > 1$. Wlog, let us assume that $D(e_1) > 1$, that is there exists $s_{11}, s_{12} \in D(e_1)$ and $s_2 \in D(e_2)$ such that $s' = s_{11} \cdot s_2$. By the definition of $D$, it holds that $S(s_{11}) = S(s_{12})$ and since the all the input-profiles in the decompisition are distinct $\mathsf{Prof}(s_{11}) \neq \mathsf{Prof}(s_{12})$. Wlog, there exists a $j \in [\ell]$ such that $\mathsf{Prof}(s_{11})_j = 0$ and $\mathsf{Prof}(s_{12})_j \in \{1, *\}$.

First, we claim that if $S(s_{11}) = S(s_{12})$ and $\mathsf{Prof}(s_{11})_j = 0$ then $\mathsf{Prof}(s_{12})_j \neq *$. By the definition of input-profiles, $S(x) \cap \mathbb{U}_j = \emptyset$ if and only if $\mathsf{Prof}(x)_j = *$. Hence, if $\mathsf{Prof}(s_{11})_j = 0$ and $\mathsf{Prof}(s_{12})_j = *$ then $S(s_{11}) \cap \mathbb{U}_j \neq \emptyset$ and $S(s_{12}) \cap \mathbb{U}_j = \emptyset$. Then, $S(s_{11}) \neq S(s_{12})$, which is a contradiction.

The remaining case is $\mathsf{Prof}(s_{11})_j = 0$ and $\mathsf{Prof}(s_{12})_j = 1$. We claim that there is no basic sub-element $s'_{11}$ of $s_{11}$ such that $S(s'_{11}) \cap \mathbb{U}_j = S^j_{k,1}$. If this not true, then $\mathsf{Prof}(s_{11}) = \perp$. Similarly, for $s_{12}$, there is no basic sub-element $s'_{12}$ such that $S(s'_{12}) \cap \mathbb{U}_j = S^j_{k,0}$. This means that $s_{11}$ and $s_{12}$ have consistently used $x_j = 0$ and $x_j = 1$ in their evaluation. Now, by Claim 6, for $S(s_{11}) = S(s_{12})$ it has to be the case that $\mathbb{U}_j \subseteq S(s_{11}) = S(s_{12})$. By Claim 10, $\mathsf{Prof}(s_{11})$ is

complete. But, multiplying an element with complete profile to another element cannot lead to any new *valid* profile. Hence, we get a contradiction to the assumption on $e'$.

**Claim 10.** *If $s$ is a single-input element such that $\mathbb{U}_j \subseteq S(s)$ for some $j \in [\ell]$, then $\mathsf{Prof}(s)$ is complete.*

*Proof.* Since $s$ is a single input element, $\mathsf{Prof}(s)_j \neq \perp$. Also, $\mathsf{Prof}(s)_j \neq *$ because $S(s) \cap \mathbb{U}_j \neq \emptyset$. Let $\mathsf{Prof}(s) = b$ for some $b \in \{0,1\}$. Also, since $\mathbb{U}_j \subseteq S(s)$, for every $i \in \mathsf{ind}(j)$ there exists a basic sub-element $s_i$ of $s$ such that $S(s_i) \cap \mathbb{U}_j = S^j_{i,b}$. Moreover, $S(s_i) = S(i, b_1, b_2)$ such that $\mathsf{Prof}(s)_{\mathsf{inp}_1(i)} = b_1$ and $\mathsf{Prof}(s)_{\mathsf{inp}_2(i)} = b_2$.

We will show that for any $k \in [\ell]$, $\mathsf{Prof}(s)_k \neq *$. By the property of dual input relaxed matrix branching program, there exists $i^* \in [n]$ such that wlog, $(\mathsf{inp}_1(i^*), \mathsf{inp}_2(i^*)) = (j, k)$. Since $\mathbb{U}_j \subseteq S(s)$, there exists a basic sub-element $s_{i^*}$ of $s$ such that $S(s_{i^*}) = S(i^*, b_1, b_2)$. Since $\mathsf{inp}_2(i) = k$, $\mathsf{Prof}(s)_k \neq *$. □

□

## 8.2 Simulation of Zero-testing

We first describe the simulation of the zero-testing at a high level and then will formally describe the simulation. The simulator uses the decomposition algorithm defined in the previous section to decompose the element $e$, that is to be zero tested, into single-input elements. Zero-testing of $e$ essentially involves zero-testing every element in its decomposition. Then we establish that if $e$ corresponds to a zero polynomial then indeed every element in the decomposition of $e$ should correspond to a zero polynomial. The intuition is that every element in its decomposition has product of $\alpha$'s which is different for every in its decomposition. And hence, with negligible probability it happens that the $\alpha$'s cancel out and yield a zero-polynomial. The only part left is to show that indeed we can perform zero-testing on every element in decomposition individually. To perform this we use the simulation algorithm defined in Section 4. We evaluate the polynomial corresponding to the single-input element on the output of the simulation algorithm. We then argue that the probability that if the single-input element was indeed a non-zero polynomial then with negligible probability the polynomial evaluates to 0. This establishes that if the polynomial is a non-zero polynomial then we can indeed detect some single-input element in its decomposition to be non-zero with overwhelming probability.

We now describe zero testing performed by the simulator $\mathsf{Sim}$. Denote the element to be zero tested to be $e$ and denote the polynomial computed by the circuit $\alpha(e)$ by $p_e$.

1. $\mathsf{Sim}$ first executes the decomposition algorithm $D$ described before on $e$. Denote the set of resulting single-input elements by $D(e)$. The output of $\mathsf{Sim}$ is either "Zero" or "Non-zero" depending on whether the element is zero or not.
2. For every $s \in D(e)$ execute the following steps:
   (a) Find the input $x$ that corresponds to the element $s$. More formally, denote $x$ by $\mathsf{Prof}(s)$. It then queries the $\mathsf{F}$ oracle on $x$ to obtain $\mathsf{F}(x)$.
   (b) Execute $\mathsf{Sim}_{\mathsf{BP}}$ on input $(1^s, \mathsf{F}(x))$, where $s$ is the size of the formula $\mathsf{F}$ to obtain the following distribution represented by the random variable $\mathcal{V}_s^{\mathsf{Sim}}$.
   $$\left\{\tilde{s}, \tilde{B}_{i,b_1^i,b_2^i}, \tilde{t} : i \in [n], b_1^i = x_{\mathsf{inp}_1(i)}, b_2^i = x_{\mathsf{inp}_2(i)}\right\}$$
   (c) We evaluate the polynomial $p_s$, which is the polynomial computed by the circuit $\alpha(s)$, on $\mathcal{V}_s^{\mathsf{Sim}}$. If the evaluation yields a non-zero result then $\mathsf{Sim}$ outputs "Non-zero".
3. For all $s \in D(e)$, if $p_s(\mathcal{V}_s^{\mathsf{Sim}}) = 0$ then $\mathsf{Sim}$ outputs "Zero".

This completes the description of the zero-testing as performed by the simulator. We now argue that the simulator runs in polynomial time.

*Running time.* From Claim 9 it follows that the first step, which is the execution of the decomposition algorithm, takes polynomial time. We now analyse the running time of the steps $(a), (b)$ and $(c)$. Step $(a)$ takes linear time. The running time of Step $(b)$ is essentially the running time of $\mathsf{Sim}_{\mathsf{BP}}$ which is again polynomial. Finally, Step $(c)$ is executed in time which is proportional to the number of queries made by the adversary to the oracle $\mathcal{O}(\mathcal{M})$ which are simulated by the simulator. Since the number of queries is polynomial, even Step $(c)$ is executed in polynomial time. Finally we argue that the Steps $(a), (b)$ and $(c)$ are executed polynomially many times. This follows from Claim 9 which shows that the number of elements in the decomposition is polynomial and hence the number of iterations is polynomial. Hence, our simulator runs in polynomial time.

We prove the following two claims about the structure of the polynomial representing the element to be zero tested that establishes the correctness of simulation. This will be useful when we will show later that element is zero iff all the elements obtained by its decomposition are zero.

**Claim 11.** *Consider an element $e$ such that $U \subseteq S(e)$. The polynomial computed by the circuit $\alpha(e)$, denoted by $p_e$, can be written as follows.*

$$p_e = \sum_{s \in D(e)} p_s = \sum_{s \in D(e)} q_{\mathsf{Prof}(s)} \cdot \tilde{\alpha}_{\mathsf{Prof}(s)}$$

*where for every $s \in D(e)$ the following holds.*

1. *The value $\tilde{\alpha}_{\mathsf{Prof}(s)}$ denotes the product $\prod_{i \in [n]} \alpha_{i,b_1^i,b_2^i}$ where $(b_1^i, b_2^i) = (\mathsf{Prof}(s)_{\mathsf{inp}_1(i)}, \mathsf{Prof}(s)_{\mathsf{inp}_2(i)})$.*
2. *$q_{\mathsf{Prof}(s)}$ is a polynomial in $\tilde{s}, \tilde{t}$ and in the entries of $\tilde{B}_{i,b_1^i,b_2^i}$. Further the degree of every variable in $q_{\mathsf{Prof}(s)}$ is 1.*

*Proof.* Consider an element $s \in D(e)$. As before denote the circuit representing $s$ by $\alpha(s)$. Alternately, we view $\alpha(s)$ as a polynomial with the $k^{th}$ monomial being represented by $s_k$. Moreover, the value $s_k$ satisfies the following three properties.

- For every $s_k$ we have that $S(s_k) = S(s)$ and therefore $U_j \subseteq S(s_k)$ for every $j \in [l]$.
- The circuit $\alpha(s_k)$ contains only multiplication gates.
- The basic sub-elements of each $s_k$ are a subset of the basic sub-elements of some $s$

From the first property and Claim 10, we have that $\mathsf{Prof}(s_k)$ is complete. Since every basic sub-element of $s_k$ is a also a sub-element of $s$ and also because $s$ is a sinlge-input element we have that $\mathsf{Prof}(s_k) = \mathsf{Prof}(s)$. Further for every $i \in [l]$, there exists a basic sub-element $e'$ of $s_k$ such that $S(e') = S(i, b_1^i, b_2^i)$ for $b_1^i = \mathsf{Prof}(s_k)_{\mathsf{inp}_1(i)}$ and $b_2^i = \mathsf{Prof}(s_k)_{\mathsf{inp}_2(i)}$. There can be many such basic sub-elements but the second property ensures that there is a unique such element. The only basic elements given to the adversary as part of the obfuscation with index set $S(i, b_1^i, b_2^i)$ are the elements $\alpha_{i,b_1^i,b_2^i} \cdot \tilde{B}_{i,b_1^i,b_2^i}$. From this it follows that we can write the polynomial $p_s$ as $q_{\mathsf{Prof}(s)} \cdot \tilde{\alpha}_{\mathsf{Prof}(s)}$ where $q_{\mathsf{Prof}(s)}$ and $\tilde{\alpha}_{\mathsf{Prof}(s)}$ are described in the claim statement. □

Before we describe the next claim we will introduce some notation. Consider a random variable $X$. Let $g$ be a polynomial. We say that $g(X) \equiv 0$ if $g$ is 0 on all the support of $X$. We define $\mathcal{V}_C^{\mathsf{real}}$ to be the distribution of the assignment of the values to $p_e$.

**Claim 12.** *Consider an element $e$. Let $p_e$ be a polynomial of degree $\mathsf{poly}(n)$ represented by $\alpha(C)$. If $p_e \not\equiv 0$ then the following holds.*

$$\mathsf{Pr}_{\mathcal{V}_C^{\mathsf{real}}}[p_e(\mathcal{V}_C^{\mathsf{real}}) = 0] = \mathsf{negl}(n)$$

*Proof.* The claim would directly follow from Schwartz-Zippel lemma if the distribution corresponding to the random variable $\mathcal{V}_C^{\mathsf{real}}$ is a uniform distribution or even if the distribution could be computed by a low degree polynomial over values uniformly distributed over $\mathbb{Z}_p$. But this is not true since the entries in $R^{-1}$ cannot be expressed as a polynomial in the entries of $R$. To this end, we do the following. We transform $p_e$ into another polynomial $p'_e$ and further transform $\mathcal{V}_C^{\mathsf{real}}$ into another distribution $\tilde{\mathcal{V}}_C^{\mathsf{real}}$ such that the following holds:

- $\underset{\mathcal{V}_C^{\mathsf{real}}}{\mathsf{Pr}}[p_e(\mathcal{V}_C^{\mathsf{real}}) = 0] = \underset{\tilde{\mathcal{V}}_C^{\mathsf{real}}}{\mathsf{Pr}}[p'_e(\tilde{\mathcal{V}}_C^{\mathsf{real}}) = 0]$
- The degree of $p'_e = \mathsf{poly}(n)$.
- The distribution corresponding to $\mathcal{V}_C^{\mathsf{real}}$ can be computed by a polynomial over values that are uniform over $\mathbb{Z}_p$.

In order to obtain $p'_e$ from $p_e$ we essentially replace the matrices $R_i^{-1}$ in $p_e$ with adjugate matrices $\mathsf{adj}(R_i) \prod_{j \neq i} \mathsf{det}(R_j)$ where $\mathsf{adj}(R_i) = R_i^{-1} \cdot \mathsf{det}(R_i)$. In a similar way we obtain $\tilde{\mathcal{V}}_C^{\mathsf{real}}$ from $\mathcal{V}_C^{\mathsf{real}}$ by replacing all the assignment values corresponding to $R_i^{-1}$ by assignment values corresponding to $\mathsf{adj}(R_i) \prod_{j \neq i} \mathsf{det}(R_j)$.

We now argue $p'_e$ satisfies all the three properties stated above. The following shows that the first property is satisfied.

$$\underset{\mathcal{V}_C^{\mathsf{real}}}{\mathsf{Pr}}[p_e(\mathcal{V}_C^{\mathsf{real}}) = 0] = \underset{\mathcal{V}_C^{\mathsf{real}}}{\mathsf{Pr}}[p_e(\mathcal{V}_C^{\mathsf{real}}) \prod_{i \in [n]} \mathsf{det}(R_j) = 0] = \underset{\tilde{\mathcal{V}}_C^{\mathsf{real}}}{\mathsf{Pr}}[p'_e(\tilde{\mathcal{V}}_C^{\mathsf{real}}) = 0]$$

We now show that the second property is satisfied. The degree of $\prod_{i \in [n]} \mathsf{det}(R_i)$ is at most $n \cdot w$ and hence the degree of $p'_e$ is at most $n \cdot w$ times the degree of $p_e$, which is still a polynomial in $n$. Finally, we show that that the third property is satisfied. To see this note that $\mathsf{adj}(R_i)$ can be expressed as polynomial with degree at most $w$ in the entries of $R_i$. Using this, we have that the distribution corresponding to $\tilde{\mathcal{V}}_C^{\mathsf{real}}$ can be computed by a polynomial (of degree at most $w$) over values that are uniform over $\mathbb{Z}_p$.

Now that we have constructed the polynomial $p'_e$, we will invoke the Schwartz-Zippel lemma on $p'_e$ to obtain the desired result as follows:

$$\underset{\mathcal{V}_C^{\mathsf{real}}}{\mathsf{Pr}}[p_e(\mathcal{V}_C^{\mathsf{real}}) = 0] = \underset{\tilde{\mathcal{V}}_C^{\mathsf{real}}}{\mathsf{Pr}}[p'_e(\tilde{\mathcal{V}}_C^{\mathsf{real}}) = 0] = \mathsf{negl}(n)$$

□

We now show that in order to zero-test an element it suffices to individually zero-test all the elements in its decomposition. This will complete the proof that our simulator satisfies the correctness property.

**Theorem 9.** *Consider an element $e$ such that $U \subseteq S(e)$ and let $p_e$ be the polynomial computed by the circuit $\alpha(e)$. We have the following:*

- *If $p_e$ is a non-zero polynomial then $p_s(\mathcal{V}_C^{\mathsf{real}}) = 0$ with negligible (in $n$) probability, for some $s \in D(e)$.*
- *If $p_e$ is a zero polynomial then $p_s(\mathcal{V}_C^{\mathsf{real}}) \equiv 0$*

*Proof.* We first consider the case when $p_e$ is a non-zero polynomial. From Claim 12, we have that $\mathsf{Pr}_{\mathcal{V}_C^{\mathsf{real}}}[p_e(\mathcal{V}_C^{\mathsf{real}}) = 0] = 0$ with negligible probability. Further since $p_e = \sum_{s \in D(e)} p_s$, we have the following.

$$\mathsf{Pr}_{\mathcal{V}_C^{\mathsf{real}}}[p_e(\mathcal{V}_C^{\mathsf{real}}) = 0] = \mathsf{Pr}_{\mathcal{V}_C^{\mathsf{real}}}[\exists s \in D(e) \ : \ p_s(\mathcal{V}_C^{\mathsf{real}}) = 0] = \mathsf{negl}(n)$$

Further We now move to the case when $p_e$ is a zero polynomial. We claim that $p_s$ is a zero polynomial for every $s \in D(e)$. From Claim 12 we know that $p_s$ can be expressed as $q_{\mathsf{Prof}(s)} \cdot \tilde{\alpha}_{i,b_1^i,b_2^i}$, where $(b_1^i, b_2^i) = (\mathsf{Prof}(s)_{\mathsf{inp}_1(i)}, \mathsf{Prof}(s)_{\mathsf{inp}_2(i)})$. Observe that the marginal distribution of $\tilde{\alpha}_{\mathsf{Prof}(s)}$ is uniform for every $s \in D(e)$. Hence, $q_{\mathsf{Prof}(s)}$ should be zero on all points of its support. In other words, $q_{\mathsf{Prof}(s)} \equiv 0$ and hence, $p_s \equiv 0$ thus proving the theorem □

As a consequence of the above theorem, we prove the following corollary.

**Corollary 3.** *Consider an element e such that $U \subseteq S(e)$ and let $p_e$ be the polynomial computed by the circuit $\alpha(e)$. We have the following.*

- *If $p_e$ is a non-zero polynomial then $p_s(\mathcal{V}_s^{\mathsf{Sim}}) = 0$ with negligible (in n) probability, for some $s \in D(e)$.*
- *If $p_e$ is a zero polynomial then $p_s(\mathcal{V}_s^{\mathsf{Sim}}) \equiv 0$.*

The proof of the above corollary follows from the above theorem and the following claim. This completes the proof of correctness of the simulation of zero-testing.

**Claim 13.** *For every single-input element s such that $U \subseteq S$ we have that the assignment $\mathcal{V}_s^{\mathsf{Sim}}$, which is the distribution output by $\mathsf{Sim}_{\mathsf{BP}}$, and the assignment to the same subset of variables in $\mathcal{V}_C^{\mathsf{real}}$ are identically distributed.*

*Proof.* The distributions of the following variables generated by $\mathsf{Sim}$ and $\mathcal{O}(\mathsf{F})$ are identical from Theorem 6:

$$R_0, \left\{ B_{i,b_1^i,b_2^i} \;\; i \in [n], b_1^i = \mathsf{Prof}(s)_{\mathsf{inp}_1(i)}, b_2^i = \mathsf{Prof}(s)_{\mathsf{inp}_2(i)} \right\}, R_n$$

Further, the following variables are sampled uniformly at random both by $\mathsf{Sim}$ and by $\mathcal{O}(\mathsf{F})$:

$$\left\{ \alpha_{i,b_1^i,b_2^i} \;:\; i \in [n], b_1^i = \mathsf{Prof}(s)_{\mathsf{inp}_1(i)}, b_2^i = \mathsf{Prof}(s)_{\mathsf{inp}_2(i)} \right\}$$

The claim follows from the fact that the assignment $\mathcal{V}_s^{\mathsf{Sim}}$ generated by $\mathsf{Sim}$ and the assignment to the same subset of variables in $\mathcal{V}_C^{\mathsf{real}}$ are both computed from the above values in the same way. □


# Acknowledgements

We thank Paul Beame and Milos Ercegovac for discussions about formula size versus depth tradeoffs for interesting functions.


# References


[ABG+13] Prabhanjan Ananth, Dan Boneh, Sanjam Garg, Amit Sahai, and Mark Zhandry. Differing-inputs obfuscation and applications. Cryptology ePrint Archive, Report 2013/689, 2013.

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

[Bar86] D A Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in $nc_1$. In *STOC*, 1986.

[BB94] Maria Luisa Bonet and Samuel R Buss. Size-depth tradeoffs for boolean formulae. *Information Processing Letters*, 49(3):151–155, 1994.

[BBC+14] Boaz Barak, Nir Bitansky, Ran Canetti, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Obfuscation for evasive functions, 2014.

[BCE91] Nader H Bshouty, Richard Cleve, and Wayne Eberly. Size-depth tradeoffs for algebraic formulae. In *FOCS*, pages 334–341. Citeseer, 1991.

[BCP13] Elette Boyle, Kai-Min Chung, and Rafael Pass. On extractability obfuscation. Cryptology ePrint Archive, Report 2013/650, 2013.

[BCPR13] Nir Bitansky, Ran Canetti, Omer Paneth, and Alon Rosen. More on the impossibility of virtual-black-box obfuscation with auxiliary input. Cryptology ePrint Archive, Report 2013/701, 2013.

[BFM14] Christina Brzuska, Pooya Farshim, and Arno Mittelbach. Indistinguishability obfuscation and uces: The case of computationally unpredictable sources. Cryptology ePrint Archive, Report 2014/099, 2014.

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *IACR Cryptology ePrint Archive*, 2001:69, 2001.

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

[Bon] `http://crypto.stanford.edu/~dabo/pubs/papers/barrington.html/`.

[BP13] Elette Boyle and Rafael Pass. Limits of extractability assumptions with distributional auxiliary input. Cryptology ePrint Archive, Report 2013/703, 2013.

[BR14a] Zvika Brakerski and Guy N Rothblum. Black-box obfuscation for d-cnfs. In *Proceedings of the 5th conference on Innovations in theoretical computer science*, pages 235–250. ACM, 2014.

[BR14b] Zvika Brakerski and Guy N. Rothblum. Virtual black-box obfuscation for all circuits via generic graded encoding. In Yehuda Lindell, editor, *TCC*, volume 8349 of *Lecture Notes in Computer Science*, pages 1–25. Springer, 2014.

[Bre74] Richard P Brent. The parallel evaluation of general arithmetic expressions. *Journal of the ACM (JACM)*, 21(2):201–206, 1974.

[CFIK03] Ronald Cramer, Serge Fehr, Yuval Ishai, and Eyal Kushilevitz. Efficient multi-party computation over rings. In Eli Biham, editor, *EUROCRYPT*, volume 2656 of *Lecture Notes in Computer Science*, pages 596–613. Springer, 2003.

[CGK14] Henry Cohn, Shafi Goldwasser, and Yael Tauman Kalai. The impossibility of obfuscation with a universal simulator. *CoRR*, abs/1401.0348, 2014.

[Cle90] Richard Cleve. Towards optimal simulations of formulas by bounded-width programs. In Harriet Ortiz, editor, *STOC*, pages 271–277. ACM, 1990.

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

[DH76] Whitfield Diffie and Martin E. Hellman. Multiuser cryptographic techniques. In *AFIPS National Computer Conference*, pages 109–112, 1976.

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

[GGH+13b] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, pages 40–49. IEEE, 2013.

[GGHR14] Sanjam Garg, Craig Gentry, Shai Halevi, and Mariana Raykova. Two-round secure mpc from indistinguishability obfuscation. 2014.

[GGHW13] Sanjam Garg, Craig Gentry, Shai Halevi, and Daniel Wichs. On the implausibility of differing-inputs obfuscation and extractable witness encryption with auxiliary input. Cryptology ePrint Archive, Report 2013/860, 2013.

[GGJ+14] Shafi Goldwasser, Vipul Gordon, Dov Goyal, Abhishek Jain, Jonathan Katz, Feng-Hao Liu, Elaine Shi, Hong-Sheng Zhou, and Amit Sahai. Multi-input functional encryption. 2014.

[GHRW14] Craig Gentry, Shai Halevi, Mariana Raykova, and Daniel Wichs. Outsourcing private ram computation. Cryptology ePrint Archive, Report 2014/148, 2014.

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

[GJKS13] Vipul Goyal, Abhishek Jain, Venkata Koppula, and Amit Sahai. Functional encryption for randomized functionalities. Cryptology ePrint Archive, Report 2013/729, 2013.

[HSW14] Susan Hohenberger, Amit Sahai, and Brent Waters. Replacing a random oracle: Full domain hash from indistinguishability obfuscation. 2014.

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

[KNY14] Ilan Komargodski, Moni Naor, and Eylon Yogev. Secret-sharing for np from indistinguishability obfuscation. Cryptology ePrint Archive, Report 2014/213, 2014.

[MO13] Antonio Marcedone and Claudio Orlandi. Obfuscation ==¿ (ind-cpa security =/=¿ circular security). Cryptology ePrint Archive, Report 2013/690, 2013.

[MR13] Tal Moran and Alon Rosen. There is no indistinguishability obfuscation in pessiland. Cryptology ePrint Archive, Report 2013/643, 2013.

[PPS13] Omkant Pandey, Manoj Prabhakaran, and Amit Sahai. Obfuscation-based non-black-box simulation and four message concurrent zero knowledge for np. Cryptology ePrint Archive, Report 2013/754, 2013.

[Spi71] Philip M Spira. On time-hardware complexity tradeoffs for boolean functions. In *Proceedings of the 4th Hawaii Symposium on System Sciences*, pages 525–527, 1971.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: Deniable encryption, and more. 2014.

[SWW99] Martin Sauerhoff, Ingo Wegener, and Ralph Werchner. Relating branching program size and formula size over the full binary basis. In Christoph Meinel and Sophie Tison, editors, *STACS*, volume 1563 of *Lecture Notes in Computer Science*, pages 57–67. Springer, 1999.