Optimizing Obfuscation: Avoiding Barrington's Theorem

Prabhanjan Ananth[†]
Department of Computer Science and Center for Encrypted Functionalities, UCLA, USA
prabhanjan.va@gmail.com

Divya Gupta[†]
Department of Computer Science and Center for Encrypted Functionalities, UCLA, USA
divyag@cs.ucla.edu

Yuval Ishai[*]
Department of Computer Science, Technion, Israel
yuvali@cs.technion.ac.il

Amit Sahai[†]
Department of Computer Science and Center for Encrypted Functionalities, UCLA, USA
sahai@cs.ucla.edu

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 Boolean formulas and branching programs – 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 $O(\ell s^{3.64})$, where s is the size of the Boolean formula to be obfuscated, and ℓ is the number of input bits to the formula. In contrast, our construction only requires the maximum number of levels of multilinearity to be roughly ℓs, or only s when considering a keyed family of formulas, namely a class of functions of the form $f_z(x) = \phi(z, x)$ where ϕ is a formula of size s. This results in significant improvements in both the total size of the obfuscation and the running time of evaluating an obfuscated formula.

Our efficiency improvement is obtained by generalizing the class of branching programs that can be directly obfuscated. This generalization allows us to achieve a simple simulation of

[*]Research supported by the European Union's Tenth Framework Programme (FP10/2010-2016) under grant agreement no. 259426 ERC-CaC, ISF grant 1361/10, and BSF grant 2012378. Research done in part while visiting UCLA and the Center for Encrypted Functionalities.

[†]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.

formulas by branching programs while avoiding the use of Barrington's theorem, on which all previous constructions relied. Furthermore, the ability to directly obfuscate general branching programs (without bootstrapping) allows us to efficiently apply our construction to natural function classes that are not known to have polynomial-size formulas.

1 Introduction

The goal of general-purpose program obfuscation is to make an arbitrary computer program "unintelligible" while preserving its functionality. Obfuscation allows us to achieve a powerful capability: software that can keep a secret. That is, software that makes use of secrets to perform its computations, but with the additional property that these secrets remain secure even if the code of the software is captured in its entirety by an adversary. At least as far back as the work of Diffie and Hellman in 1976 [24][1], researchers have contemplated applications of general-purpose obfuscation. Indeed, if secure general-purpose obfuscation could be cryptographically achieved *efficiently*, the implications to computer security would be profound [5].

To understand why obfuscation can be so useful, it is instructive to contemplate what kinds of secrets we might want to hide within our software code. An important instance of such secrets is hiding the existence and nature of *rare input/output behavior* that our software may exhibit. This leads to several interesting motivating scenarios:

- Our software may be a control algorithm that is programmed to enter a failsafe mode on certain rare and hard-to-predict inputs. We would not want an adversary that gains access to the code of the control software to be able to learn these rare inputs. By securely obfuscating the control software, the existence of the failsafe mode itself would be hidden from the adversary.
- We may *modify* software to introduce such rare input/output behavior to suit our goals. Consider the problem of software watermarking, where we want to add an undetectable imprint to our software that we can later identify. We may do so by modifying the behavior of our software, so that on several rare and hard-to-predict inputs, it outputs a watermark code instead of performing its usual computation. An obfuscated version of this modified software would hide the existence of these imprints, and thereby also prevent an adversary from removing them unless the adversary rewrites from scratch almost all of the software.
- So far our examples have dealt with hiding known rare input/output behavior. But obfuscation could also be used to hide the existence of *unknown* and unintentional rare input/output behavior: Consider software bugs that are particularly resistant to good-faith software testing, because the input/output behavior that is affected by these bugs only arises from inputs that are rare and hard to predict given only the functionality of the software. Then, obfuscation can be used to hide the existence of such software bugs (and the vulnerabilities they introduce), even from an attacker that has the code of the software.
- Finally, turning the previous example around, obfuscation can also be used to hide which of these software bugs are being fixed by a software patch, thereby preventing adversaries from learning vulnerabilities from software patches and using this knowledge to attack unpatched software.

As these motivating scenarios illustrate, secure obfuscation would greatly expand the scope of security problems addressable through cryptographic means. However, efficient and secure obfuscation would also have powerful applications to data security, specifically to protecting

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

against data breaches by low-level insiders. Low-level insiders can cause data breaches if they go rogue, or if their computing systems are compromised through theft or malware attack. As a result, a critical problem arises when such insiders hold decryption keys – indeed even low-level insiders may need such keys to perform basic functions. For example, an employee tasked with generating summaries of financial statistics may need decryption keys in order to decrypt sensitive financial spreadsheets. If this decryption key is captured by an adversary, however, it can be used to steal vast quantities of sensitive information, even though the decryption key was only meant to allow the insider to generate low-value statistical summaries. Obfuscation, however, provides a powerful solution to this problem: The decryption keys can be safely hidden within the statistical summary generation software that is entrusted to the low-level insider. Then, even if the insider turns rogue, the only power he can derive from his software is the ability to generate statistics[2]; he cannot abuse his position to directly decrypt the underlying financial files.

The above examples provide only a fractional view of the applicability that efficient secure obfuscation would have to computer security. However, until 2013, even heuristic constructions for secure general-purpose obfuscation were not known.

This changed with the work of Garg, Gentry, Halevi, Raykova, Sahai, and Waters [28], which gave the first candidate cryptographic construction for a general-purpose obfuscator. Formal exploration of the applications of general-purpose obfuscation began shortly thereafter [28, 49]. Since then, the floodgates have opened, and many new applications of general-purpose obfuscation have been explored [10, 13, 44, 12, 1, 42, 7, 34, 9, 45, 29, 32, 3, 17, 39, 30, 35, 14, 15, 27, 21].

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 [4], building upon [28, 15]. However, the general-purpose obfuscator presented in [4] (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 [28] 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 (equivalently, $\mathbf{NC}^1$ circuits), building upon a simplified subset of the Approximate Multilinear Maps framework of Garg, Gentry, and Halevi [26] 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 works of [28, 15, 4] all adopt a method for bootstrapping using Fully Homomorphic Encryption. This bootstrapping method works provably with the security definition of indistinguishability obfuscation, and can rely on well-studied cryptographic assumptions such as the LWE assumption. Alternatively, the earlier work of Goyal et al. [33] constructed a universal stateless hardware token for obfuscation that can be implemented by polynomial-size boolean formulas using a pseudorandom function in $\mathbf{NC}^1$. Applebaum [2] gives a simpler alternative construction that has the disadvantage of requiring the size of the Boolean formulas to be polynomial in the input size and the security parameter (rather than only in the security parameter in [33]). Using either of these alternative approaches [33, 2], however, requires an ad-hoc (but arguably plausible) assumption to bootstrap from obfuscation for Boolean formulas to obfuscation for general circuits.

[2]Of course, the statistical software itself must be carefully written to avoid vulnerabilities that allow a user to extract specific sensitive information by making unexpected statistical queries.

Our work focuses on improving the efficiency of the first of these steps: namely, the core obfuscator for Boolean formulas. We give one set of results for the setting of boolean formulas over the {AND, NOT, OR}-basis, and another set of results for general basis.

Previous constructions of a core obfuscator [28, 15, 4] first apply Barrington's theorem [6] to convert the Boolean formula into an equivalent "matrix branching program," which is then obfuscated. Roughly speaking, a matrix branching program computes an iterated product of n full-rank matrices, where each matrix in the product is determined by one of the input bits, and the result of the product should be either the identity matrix (corresponding to an output of 1) or some other fixed full-rank matrix (corresponding to an output of 0). The length of the program is n and its width is the matrix dimension.

For any circuit or formula of depth d, Barrington's theorem gives a constant-width matrix branching program of length 4^d. Since the length is exponential in the formula depth, it is crucial to balance the depth of the formula in order to avoid the exponential blowup. 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 $1.73 \log s + d_0$ by Khrapchenko [37, 36] where d_0 is a constant. However, the constant d_0 is quite large, which can have an adverse effect on concrete efficiency.[3] Instead, one can balance the formula using a method by Preparata and Muller [48]. The depth of the balanced formula obtained by this method is $1.82 \log s$. There have been other works which try to optimize the *size* of balanced formulas [11], but the depth of the formula obtained by these works is worse.

The matrix branching program obtained by applying Barrington's theorem to a formula of depth $1.82 \log s$ has length $s^{3.64}$. This is a major source of inefficiency. In particular, the bound of $s^{3.64}$ on the length of the branching program not only affects the number of elements given out as the final obfuscation, but also the number of levels of multilinearity required by the scheme. Since the size of each multilinear encoding grows with the number of levels of multilinearity required in known realizations of approximate multilinear maps [26, 22], 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, as well as the need to balance Boolean formulas at all. In fact, this strategy can be efficiently applied to general (deterministic or even non-deterministic) *branching programs*, which are not known to be simulated by polynomial-size formulas. Our strategy employs variants of randomization techniques that were used in the context of secure multiparty computation [25, 23], adapting them to the setting of obfuscation.

A crucial first step is to formulate a notion of a "relaxed matrix branching program" (RMBP) which relaxes some of the requirements of matrix branching programs needed in [28, 15, 4]. The relaxation replaces permutation matrices by general full-rank matrices over a finite field and, more importantly, determines the output by testing whether some fixed entry in the matrix product is nonzero. (See Section 2.3 for a formal definition.) We show how to adapt the construction and security proofs of [4] to work with RMBPs. The efficiency of this obfuscation will be discussed in more detail below. Roughly speaking, given the efficiency of current candidate multilinear encodings, the complexity of obfuscating RMBPs grows quadratically with the width and cubicly with the length. For now, we will measure efficiency in terms of the length and width of the RMBP.

Armed with the ability to obfuscate RMBPs, we look for simple and efficient ways to convert Boolean formulas and traditional types of branching programs into RMBPs without invoking Barrington's theorem. For this, we can use a previous transformation implicit in [25] towards

[3] Note that once we apply Barrington's theorem, d_0 goes into the exponent and hence the size of the resulting obfuscation scheme will incur a factor of 4^{d_0}.

converting any ordinary graph-based non-deterministic branching program[4] of size s into an RMBP of length s and width $2(s+1)$. We also provide more efficient variants of this transformation that apply to classes of layered branching programs that satisfy certain technical conditions and arise in natural applications.

The above is already enough for efficiently obfuscating functions that are represented by small branching programs. However, in many cases functions are more naturally represented by Boolean formulas. In order to efficiently obfuscate formulas, we turn to the (abundant) literature on simulating formulas by branching programs. In the case of formulas consisting of only AND, OR, and NOT gates, we can use a simple transformation of any such formula of size s into a branching program of the same size (cf. Theorem 6 in [43] and Appendix B).

The above simple transformation is limited in that it does not directly apply to formulas with XOR gates, and even without such gates its efficiency leaves much to be desired. Concretely, a formula of size s is transformed into an RMBP whose length and width are roughly s and $2s$, respectively, leading to a total of $O(s^3)$ matrix elements. To get around both limitations we rely on the work of Giel [31], which builds on previous results of [11, 20] to efficiently transform a formula over the full basis to a layered branching program of constant width. The layered branching program described in [31] satisfies our conditions and can be used to obfuscate formulas over the full set of binary gates. Concretely, a formula of size s can be transformed into an RMBP of length $O(s^{1+\epsilon})$, for an arbitrarily small constant $\epsilon > 0$, and constant width (depending only on ϵ).

As in previous obfuscation techniques [28, 15, 4], a direct application of the above methods reveals the order in which input variables are read. Thus, to obfuscate a class of branching programs or formulas which may read the inputs in a varying order, we (as well as previous works) need to apply an additional step to make the RMBP family *input-oblivious*. This incurs an additional multiplicative overhead of ℓ to the length. However, this step and the resulting overhead can be avoided when the RMBP family is already input-oblivious. This is guaranteed in the useful case of obfuscating a class of *keyed* functions, namely a class of functions of the form $f_z(x) = \phi(z, x)$ where ϕ is a publicly known branching program or formula of size s. In other words, the goal is to obfuscate the class $\phi(z, \cdot)$ to hide the key z. In this case, an RMBP for ϕ can be easily turned into an input-oblivious family of RMBPs for the class f_z with no additional overhead.

Efficiency comparison. We now quantify the efficiency improvements we obtain over previous work; we will do so both asymptotically and with explicit numbers through examples. The efficiency of our obfuscation scheme can be compared to previous ones by considering (1) the level κ of the multilinear encoding being employed, and (2) the number S of encoded field elements. The parameter κ is of special importance, as the bit-length of each encoded element in current multilinear encoding candidates [26, 22] grows quadratically with κ. Thus, a good estimate for the total size (in bits) of an obfuscated program is $\tilde{O}(\kappa^2 \cdot S)$, where $\tilde{O}$ hides a multiplicative factor which depends polynomially on the security parameter. Moreover, our constructions (as well as previous ones) can be implemented so that the running time required for evaluating the obfuscated program is quasi-linear in the obfuscation size. Thus, from here on we will not explicitly refer to the asymptotic running time.

The concrete cost of implementing optimized multilinear encoding candidates is a subject of much ongoing research [22, 40, 41], and as of the time of this writing, explicit running time and size estimates for multilinear candidates optimized for obfuscation[5] are not available for the κ

[4] A non-deterministic branching program is a standard computational model that corresponds to non-deterministic logarithmic space. Such branching programs are believed to be strictly stronger than deterministic branching programs and formulas (see below) and strictly weaker than general circuits. See Section 2.2 for a formal definition.

[5] We note that obfuscation only requires Multilinear Jigsaw Puzzles [28], a strict subset of the full multilinear map functionality, which allows for substantial efficiency improvements in implementations. However, as of this writing, no experimental study of Multilinear Jigsaw Puzzle implementations has been completed.

values that we need. However, as research in this direction is still in its infancy, it is reasonable to expect major improvements in the near future. For this reason, we do not attempt to provide real-life running time estimates, but rather compare our constructions with previous ones by considering the parameters κ and S described above.

The obfuscation methods from [28, 15, 4], when applied to a (strict) matrix branching program of length n and width w (one whose evaluation involves the product of n matrices of size $w \times w$) requires $\kappa = n$ levels of multilinearity and $S = w^2 n$ encoded elements. The same holds for our method when applied to an RMBP of length n and width w. Our simple and direct transformation for a (keyed) formula of size s over the standard basis yields an RMBP of length $n \approx s$ and width $w \approx 2s$. This should be compared with the previous Barrington-based solution combined with the best known formula balancing results, leading to a matrix branching program with parameters $n = O(s^{3.64})$ and $w = O(1)$. Thus, under the quadratic cost assumption mentioned above, the obfuscation size is improved from $\tilde{O}(\kappa^2 \cdot S) = \tilde{O}(s^{10.92})$ to $\tilde{O}(s^5)$. (For the case of a completely balanced formula, the obfuscation size of the previous method is reduced to $\tilde{O}(s^6)$.) By further applying the result from [31], we can obfuscate formulas over a full basis while reducing the total size to $\tilde{O}(s^{3(1+\epsilon)})$. See Table 1 for a detailed summary of old and new results for obfuscating formulas.

We note that even if future implementations of multilinear maps achieve an encoding size that only grows linearly with κ, our results would still yield significant improvements. (An encoding size that grows sublinearly with κ seems out of reach with current lattice-based methods, due to error growth.)

Finally, to the best of our knowledge, it is not known how to simulate general branching programs (even deterministic ones) by strict (i.e., non-relaxed) matrix branching programs with a polynomial overhead. Thus, for the purpose of obfuscating branching programs without the use of bootstrapping, our method provides a *super-polynomial* efficiency improvement over previous core obfuscators. See Table 2 for a detailed summary of old and new results for obfuscating branching programs.

Examples. We illustrate our concrete efficiency gains by two examples. The first example is motivated by the goal of obfuscating a psudorandom function (PRF) and deals with a conjectural PRF. As discussed above, PRFs can be used to bootstrap general obfuscation [33, 2]. While practical PRF candidates such as AES are not known to have small formulas or branching programs, it seems plausible that there are good PRF candidates with relatively small formulas or layered non-deterministic branching programs. Suppose that a PRF family $f_z : \{0,1\}^{100} \to \{0,1\}$ can be implemented by a layered, invertible non-deterministic branching program of length 300 and width 30 (see Section 2.2 for definition). Obfuscating such a PRF family using our methods would require roughly 270,000 encoded field elements, with multilinearity $\kappa \approx 300$. In contrast, obfuscating such a PRF with previous approaches would require one to decide reachability in a layered graph of length 300 and width 30. The latter can be done using at least $\lceil \log_2 300 \rceil = 9$ levels of recursion, each implemented by a circuit of depth 6, leading to a circuit of at least depth 54. Thus, a direct use of the Barrington-based approach would require using $\kappa > 2^{100}$ levels, which is infeasible. We pose the design and analysis of such an "obfuscation-friendly" PRF as a major open question that is motivated by our work.

As another example, consider the task of obfuscating a "fuzzy match" functionality, defined by a Hamming ball of radius r around a secret point $z \in \{0,1\}^n$. That is, the obfuscated function $f_z(x)$ evaluates to 1 if the Hamming distance between x and z is at most r, and evaluates to 0 otherwise. Functions from this class can be implemented by (input-oblivious, special) layered branching programs of width $r+1$ and length n, leading to an obfuscation that contains roughly $4r^2 n$ encoded elements with multilinearity $\kappa \approx n$. For the case $n = 100$ and $r = 20$, we get an obfuscation that consists of roughly 160,000 encoded elements, with multilinearity $\kappa \approx 100$. In contrast, representing such functions by formulas or low-depth circuits, which is essentially equivalent to computing the (n, r)-threshold function, leads to a best known formula size $s > n^{4.4}$

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

Work	Levels of Multilinearity	Size of Obfuscation/ Evaluation Time
[4] + [48] (previous work) $\{\text{AND}, \text{OR}, \text{NOT}\}$-basis	$O(s^{3.64})$	$O(s^{3.64}) \cdot \tilde{O}((s^{3.64})^\phi)$
This work (direct) $\{\text{AND}, \text{OR}, \text{NOT}\}$-basis	s	$4s^3 \cdot \tilde{O}(s^\phi)$
This work + [31] any complete basis	$O(s^{1+\epsilon})$	$O_\epsilon(s^{(1+\epsilon)}) \cdot \tilde{O}((s^{(1+\epsilon)})^\phi)$

and circuit depth $d > 4.9 \log_2 n$ [47, 51], which in turn require $\kappa > 10^{19}$ levels of multilinearity using previous obfuscation methods. Thus in this concrete example, our improvement just to the level of multilinearity κ is over 10^{17}; the improvement in the overall running time and size would be even greater.

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 [4] does: by showing that our constructions unconditionally achieve a strong virtual black-box notion of security [5], against adversaries that are limited to algebraic attacks allowed in a generic multilinear model. In fact, our obfuscators are information-theoretically secure against query-bounded adversaries in this generic model. We note that our work actually provides a new *feasibility* result in the generic multilinear model, namely an information-theoretic (and unconditional) obfuscation for non-deterministic branching programs which capture the complexity class NL. This should be compared to previous results in the same model, which only efficiently apply to formulas (or the complexity class NC^1).

As in the case of [4], our security proof in the generic model can be interpreted in two natural ways: (1) Our proof can be viewed as evidence of virtual black-box security for practical applications, in a similar spirit to proofs of security in the random oracle model [8]. It is important to note that analogous to known attacks on contrived schemes in the random oracle model (e.g. [19]), there are known attacks to virtual black-box security for obfuscating quite contrived functionalities [5]. However, no attacks are known for virtual black-box obfuscation for obfuscating practical functionalities. (2) Our proof can also be viewed as evidence that our obfuscator achieves the notion of indistinguishability obfuscation [5], which is a definition of security of obfuscation that does not suffer from any known attacks even for contrived functionalities, but which nevertheless has proven to be quite useful.

Organization. In Section 2 we define virtual black box obfuscation and various notions of branching programs relevant to our construction. Then in Section 3 we give a general transformation from a branching program to a relaxed matrix branching program (RMBP). For

Table 2: Comparing the efficiency of obfuscation schemes for keyed non-deterministic branching programs and special layered branching programs, as defined in Section 2.2. For a general branching program, s denotes the size of the branching program. For a special layered branching program, n is the length and w is the width. Other notation is as in Table 1.

Work	Levels of Multilinearity	Size of Obfuscation/ Evaluation Time
Previous work (general)	$s^{O(\log s)}$	$s^{O(\log s)} \cdot \tilde{O}(s^{O(\log s)})$
This work (general)	s	$4s^3 \cdot \tilde{O}(s^{\phi})$
Previous work (special layered)	$n^{O(\log w)}$	$n^{O(\log w)} \cdot \tilde{O}(n^{O(\log w)})$
This work (special layered)	n	$4nw^2 \cdot \tilde{O}(n^{\phi})$

completion, we show the transformation from formulae to branching programs in Appendix B. The next step is randomization of these RMBPs which is described in Section 4. Finally, we construct an obfuscation scheme by showing how to encode the elements in the randomized RMBP using ideal multilinear encodings (defined in Section 5) and the concept of straddling sets (defined in Section 6). The construction of the obfuscation scheme is described in Section 7.

2 Preliminaries

We denote the security parameter by λ. We use $[n]$ to denote the set $\{1, \ldots, n\}$.

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

Let $\mathcal{M}$ be some oracle. Below we define "Virtual Black-Box" obfuscation in the $\mathcal{M}$-idealized model taken verbatim from [4]. In this model, both the obfuscator and the evaluator have access to the oracle $\mathcal{M}$. However, the function family that is being obfuscated does not have access to $\mathcal{M}$.

Definition 1. *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 Sim, and a negligible function μ such that for all PPT distinguishers D, for every $\ell \in \mathbb{N}$ and every $C \in \mathcal{C}_\ell$:*

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

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

For definition of Boolean formulas see Appendix A. Next, we describe branching programs.

2.2 Branching Programs

In this section we define a non-deterministic branching program, and several types of layered branching programs that are useful for our purpose.

A *non-deterministic branching program* (BP) is a finite directed acyclic graph with two special nodes, a source node and a sink node, also referred to as an "accept" node. Each non-sink node is labeled with a variable x_i and can have arbitrary out-degree. [6] Each of the out-edges is either labeled with $x_i = 0$ or $x_i = 1$. The sink node has out-degree 0. In the following, we denote a branching program by BP and denote the restriction of the branching program consistent with input x by $\mathsf{BP}|_x$. An input $x \in \{0,1\}^\ell$ is accepted if and only if there is a path from the source node to the accept node in $\mathsf{BP}|_x$. Note that an input can have multiple computation paths in $\mathsf{BP}|_x$. The *length* of the BP is the maximum length of any such path in the graph. The *size* s of the branching program is the total number of non-sink nodes in the graph, i.e., total number of nodes minus 1.

A *layered* branching program is a branching program such that nodes can be partitioned into a sequence of layers where all the nodes in one layer are labeled with the same variable and edges go only from nodes of one layer to the next. We can assume without loss of generality that the first layer contains the source node and the last layer contains the sink node. The *length* n of a layered branching program is the number of layers minus 1 and its *width* w is the maximum number of nodes in any layer. It will be convenient to assume that a layered BP has exactly w nodes in each layer. We denote the k^{th} node in layer i by $v_{i,k}$ for $0 \le i \le n$ and $k \in [w]$.

The following nonstandard types of branching programs will be useful for our purposes. A *special* layered branching program is a layered branching program with the following additional property. For each layer i, $0 \le i < n$, and each $k \in [w]$, there is an edge from $v_{i,k}$ to $v_{i+1,k}$ labeled by both 0 and 1 (namely, this edge is consistent with all inputs).

Finally, we define an *invertible* layered branching program as follows. An invertible layered branching program is a type of a layered branching program. Corresponding to each $i \in [n]$, we define two $w \times w$ matrices $B_{i,0}$ and $B_{i,1}$ as follows: $B_{i,b}[x,y] = 1$ if and only if there is an edge from node $v_{i-1,x}$ to node $v_{i,y}$ labeled with b. Otherwise, $B_{i,b}[x,y] = 0$. We say that the layered branching program is *invertible* if $B_{i,b}$ is full rank for all $i \in [n]$ and $b \in \{0,1\}$.

2.3 Relaxed Matrix Branching Programs

In this section we define the original notion of matrix branching programs used in [28] followed by our notion of relaxed matrix branching programs.

Definition 2 (Matrix Branching Program (MBP))**.** *[4] A matrix branching program of width w and length n for ℓ-bit inputs is given by a $w \times w$ permutation matrix P_{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 BP. The output of BP on input $x \in \{0,1\}^\ell$, denoted by $\mathsf{BP}(x)$, is

[6] We assume for simplicity that the out-degree is bound by some fixed constant (say 4), so that the total number of paths is bounded by $2^{O(s)}$ as opposed to s^s.

determined as follows:

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

We say that a family of MBPs are input-oblivious *if all programs in the family share the same parameters w, n, ℓ and the evaluation function* inp.

Barrington [6] showed that every circuit with depth d and fan-in 2 can be represented by a MBP of length at most 4^d and width 5. Previous works [28, 15, 4] used MBPs obtained by applying Barrington's theorem to obfuscate circuits. Since the MBP obtained has length exponential in the depth of the circuit, this turns out to be a bottleneck for efficiency. In this work, we will use relaxed MBPs towards obfuscation.

In MBP after evaluation we either get $I_{w\times w}$ or P_{reject} which decides the output. We relax this requirement as follows. We only require that a single designated entry in the final product is either 0 or non-zero depending on the output and place no restriction on other entries. Note that this is a further relaxation of the notion considered in [46]. More formally, we define the notion of relaxed matrix branching programs as follows.

Definition 3 (Relaxed MBP (RMBP))**.** *Let R be any finite ring. A relaxed matrix branching program (over R) of size w and length n for ℓ-bit inputs is given by a sequence:*

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

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

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

Dual-input Relaxed Matrix Branching Programs. Similar to [4], for the proof of obfuscation we would need to consider dual input matrix branching programs. We define dual input RMBP as follows.

Definition 4 (Dual Input RMBP)**.** *Let R be a finite ring. A dual-input relaxed matrix branching program (over R) of size w and length n for ℓ-bit inputs is given by a sequence:*

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

where each B_{i,b_1,b_2} is a $w \times w$ full-rank matrix and $\mathsf{inp}_1, \mathsf{inp}_2 : [n] \to [\ell]$ are the evaluation functions of BP. *The output of* BP *on input $x \in \{0,1\}^\ell$, denoted by $\mathsf{BP}(x)$, is determined as follows:*

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

We say that a family of matrix branching programs is input-oblivious *if all programs in the family share the same parameters w, n, ℓ and the evaluation functions $\mathsf{inp}_1, \mathsf{inp}_2$.*

For the purpose of obfuscation we would consider *dual input oblivious relaxed matrix branching programs.*

3 From BP to Relaxed Matrix BP

In this section we describe a sequence of transformations which allow us to transform a non-deterministic branching program of size s to a relaxed matrix branching program of width $2(s+1)$ and length s. These transformations are close variants of similar transformations from [25]. The main steps are to convert a non-deterministic branching program to a special layered branching program, then to an invertible layered branching program, and finally to an RMBP. These intermediate steps can be independently useful, as they allow for more efficient transformations of special or invertible layered branching programs into RMBPs.

Branching program to special layered branching program.

Lemma 1. *Any non-deterministic branching program* BP *of size s can be efficiently converted to an equivalent special layered branching program* SLBP *of length s and width $s+1$.*

Proof Sketch. Recall that since we do not include the sink node in the size of a BP, a BP of size s has $s+1$ nodes. Given a branching program with $s+1$ nodes, first do a topological sort of the nodes, say $\{v_1, \ldots, v_{s+1}\}$. Without loss of generality, assume that v_1 is the source node and v_{s+1} is the sink node. We construct a special layered branching program with $s+1$ layers where each layer has $s+1$ nodes as follows. Let the nodes in layer i be $\{v_{i,1}, \ldots, v_{i,s+1}\}$. That is, we denote k^{th} node in layer i by $v_{i,k}$. For each $0 \leq i < s$ we do the following: Let node v_{i+1} be labeled with x_j in original BP. Then, we label layer i with x_j. We draw the outgoing edges from node $v_{i,i+1}$ to node $v_{i+1,k}$ if there was an edge from v_{i+1} to v_k in the original BP. Labels on the edges are retained. Now, we also add edges between $v_{i,k}$ to $v_{i+1,k}$ for all $k \in [s+1]$ for both $x_j = 0$ and $x_j = 1$.

It is easy to see that there is a path between source, i.e., v_1 to the accept node, i.e., v_s in the original branching program if and only if there is a path from $v_{0,1}$ to $v_{s,s+1}$.

Special layered branching program to an invertible layered branching program.

Lemma 2. *Any special layered branching program* SLBP *of length n and width w can be efficiently converted to an equivalent invertible layered branching program* ILBP *of length n and width $2w$.*

Proof Sketch. Let edges in layer i be $\{v_1, \ldots, v_w\}$. For each layer i, where $0 \leq i < n$, we add w dummy nodes, say $\{v_{i,w+1}, \ldots, v_{i,2w}\}$ and add the following edges. For each layer i and $j \in [w]$ add edges from $v_{i,j}$ to $v_{i+1,w+j}$ and from $v_{i,w+j}$ to $v_{i+1,j}$.

It is easy to see that the new layered branching program is invertible. More precisely, the columns can be re-arranged so that the matrices obtained are upper-triangular with all the main diagonal entries set to 1. It also easy to observe that if $\mathsf{SLBP}(x) = 1$ then $\mathsf{ILBP}(x) = 1$. For the other direction, note that adding these extra nodes and edges does not create a path between any two original nodes if there was no path before.

Invertible layered branching program to relaxed matrix branching program.

Lemma 3. *Any invertible layered branching program* ILBP *of length n and width w can be efficiently converted to an equivalent relaxed matrix branching program* RMBP *of length n and width w.*

Proof. Consider a large enough [7] prime p. Corresponding to each layer in the layered branching program, we will have two $(w \times w)$ matrices $B_{i,0}$ and $B_{i,1}$ over $\mathbb{Z}_p$. Let the label of this layer be x_j for some $j \in [\ell]$. For $i \in [n]$ and $b \in \{0,1\}$, define $B_{i,b}$ as follows:

[7] In the following construction, we use the fact that the prime p is large enough so that there are no wrap-arounds while multiplying the matrices. In particular, assume that $p = 2^{\Omega(n)}$.

For any $x, y \in [w]$, set $B_{i,b}[x, y] = 1$ if there is an edge between $v_{i-1,x}$ to $v_{i,y}$ labeled $x_j = b$. Set the rest of the entries in $B_{i,b}$ to be 0. We define inp to be a function from $[n]$ to ℓ, where ℓ is the input length of ILBP. We set $\mathsf{inp}(i) = j$ if all the nodes in the i^{th} layer depend on the j^{th} input bit.

Since we are given an invertible layered branching program, it is easy to see that all the matrices are full-rank. Without loss of generality, let the source node be the node $v_{0,1}$ and accept node be the node $v_{n,w}$ of the invertible layered branching program. Then,

Claim 1. *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 $\mathsf{ILBP}(x) = 1$.*

Proof Sketch: We prove this via induction on the number of layers in the branching program. Intuitively, we will prove that the following invariant is maintained. Let $P_j = \prod_{i=1}^{j} B_{i,x_{\mathsf{inp}(i)}}$. Then, $P_j[x, y]$ will denote the number of paths from $v_{0,x}$ to $v_{j,y}$. In particular, $P[1, w]$ captures the number of paths from the source node to the sink node. And hence, $P[1, w]$ is non-zero iff $\mathsf{ILBP}(x) = 1$.

We argue this by induction. We define graph G_j, for $j \in [n]$, to be a subgraph of $\mathsf{ILBP}|_x$ 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 $\mathsf{ILBP}|_x$ 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 $\mathsf{ILBP}|_x$ 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 $\mathsf{ILBP}|_x$ and more specifically, $P[1, w]$ is non-zero iff $\mathsf{ILBP}(x) = 1$. This proves the lemma.

□

Theorem 1. *Any non-deterministic branching program BP of size s can be efficiently converted to an equivalent relaxed matrix branching program RMBP of length s and width $2(s + 1)$.*

Proof. It follows directly from Lemmas 1, 2 and 3. □

Converting relaxed matrix branching program to dual input oblivious relaxed matrix branching program. First note that if the family of invertible layered matrix branching programs is input oblivious, then the relaxed matrix branching program obtained from the above transformation would also be input oblivious. If that is not the case, we can convert it to a dual-input relaxed matrix branching program by incurring a multiplicative cost of ℓ in the length of the branching program. More formally,

Lemma 4. *Any relaxed matrix branching program $\mathsf{RMBP} = (\mathsf{inp}, B_{i,0}, B_{i,1})_{i\in[n]}$ of length n and width w can be efficiently converted to a dual-input oblivious relaxed matrix branching program of length at most $n\ell$ and width w.*

Proof. We first make our relaxed matrix branching program oblivious, i.e. make the evaluation function 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 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 ℓ 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 inp_2 as follows: Consider i of the form $k_1\ell(\ell-1) + k_2\ell + k_3$ then

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

□

3.1 From Formula to RMBP

Direct construction. Transforming formulas to branching programs is a well studied problem [43, 6, 20, 50]. In particular, it is well known [43] that formula of size s over AND, OR, and NOT gates can be converted to a branching program of essentially the same size; for self containment, we describe such a transformation in Appendix B which satisfies the following lemma.

Lemma 5. *Any formula of size s can be converted to a branching program of size s.*

Using the transformations described in the previous section, we obtain the following.

Theorem 2. *Any formula of size s can be efficiently converted to an equivalent relaxed matrix branching program of width $2(s+1)$ and length s. Moreover, it can be converted to a dual input oblivious matrix branching program of width $2(s+1)$ and length $s\ell$.*

Keyed formulas. Consider the class of keyed formulas, namely a class of formulas of the form $f_z(x) = \phi(z,x)$ such that ϕ is a formula of size s. While obfuscating this class of formulas, we only need to hide the key z since ϕ is public. Since we do not require the matrix branching program to be input oblivious, we do not incur the additional factor of ℓ in the length of the matrix branching program. So the length of the branching program for this class of functions is at most s.

Alternate approach. We note that there is an asymptotically more efficient transformation to obtain relaxed matrix branching program using the work of Giel [31]. The transformation consists of the following steps – first the formula is balanced and then the resulting balanced formula is converted to a linear bijection straightline-program (LBSP) which is then converted to an RMBP. More formally, we have the following result due to Giel [31]. For completeness sake, we present the proof of the theorem in Appendix C.

Theorem 3. *[31] Given a boolean formula of size s over any complete basis, there exists a relaxed matrix branching program of size $O(s^{1+\epsilon})$ with the width of each matrix is a constant depending only on ϵ,where $\epsilon > 0$ is any constant.*

4 Randomization of RMBP

In this section, we describe how to randomize the matrices in the (dual-input and oblivious) relaxed matrix branching program obtained from the construction in Section 3. The result of the randomization process is another relaxed matrix branching program such that the restriction of the relaxed matrix branching program[8] on input x can be simulated by just knowing the output of the branching program on input x. Looking ahead, this property will come in handy when proving the security of the obfuscation scheme in the ideal graded encoding model. The randomization technique we employ closely follows a similar randomization technique that was used in [23] in the context of secure multiparty computation.

The non-triviality of the randomization process here compared to [4] is the following: in [4] the product matrix corresponding to an input x depends only on the output of the function on input x. More specifically it is either an identity matrix or a fixed matrix P_{reject} (Definition 2). Thus, the product matrix does not reveal any information about the branching program. However, in our case the entries in the product matrix might contain useful information about the branching program – specifically the product matrix in our RMBP captures the number of paths between every pair of vertices. Hence, we have to randomize the matrices in such a way that the product of the matrices only reveals information about the output of the function. We do this in two steps. In the first step we design a randomization procedure, denoted by randBP, which reveals just the $(1,w)^{th}$ entry of the product matrix. Note that this itself will not be enough for us because the $(1,w)^{th}$ entry essentially contains the number of paths between the source and the accept vertex and hence has more information than just the output of the function. And so in the second step, we describe how to randomize the RMBP using the procedure randBP', such that the resulting (equivalent) RMBP when restricted to any particular input x can be simulated by just knowing the output of the RMBP on x.

We first describe the randBP procedure. Though the procedure randBP to randomize our matrices is similar in spirit to Kilian's randomization (also used in [28, 4]), the way we will simulate these matrices will deviate from that of Kilian.

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 ℓ 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 convenience, let e_1 be a row vector and e_w we a column vector.

Procedure 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 ℓ bits.
Procedure $\mathsf{randBP}(\mathsf{BP})$:

- Pick $n+1$ random full-rank matrices $R_0,\ldots,R_n \in \mathbb{Z}_p^{w\times w}$.

[8] Recall that the restriction of a relaxed matrix branching program $\mathsf{BP} = (\mathsf{inp}_1, \mathsf{inp}_2, \{B_{i,b_1,b_2}\}_{i\in[n],b_1,b_2\in\{0,1\}})$ is defined to be $\{B_{i,x_{\mathsf{inp}_1(i)},x_{\mathsf{inp}_2(i)}}\}$.

- 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 $\widetilde{\mathsf{BP}} = \left(\mathsf{inp}_1, \mathsf{inp}_2, \tilde{s}, \left\{\tilde{B}_{i,b_1,b_2}\right\}_{i\in[n],b_1,b_2\in\{0,1\}}, \tilde{t}\right)$.

It follows that the branching program output by the above procedure on input BP is functionally equivalent to BP.

We can construct a simulator $\mathsf{Sim}_{\mathsf{BP}}$ such that the following theorem holds. At a high level, the theorem states that the matrices in $\widetilde{\mathsf{BP}}$ (which is the output of randBP on BP) when restricted to a particular input x can be simulated by just knowing the $(1,w)^{th}$ entry in the product matrix obtained by evaluating BP on input x.

Theorem 4. *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, of width w and input of ℓ bits. Then for every $x \in \{0,1\}^\ell$,*

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

Proof. We first describe the simulator $\mathsf{Sim}_{\mathsf{BP}}$ which simulates the output of 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 $(1^n, 1^w, 1^\ell, P_x[1,w])$ 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.

Theorem 5. *([38]) Consider a dual-input branching program* $\mathsf{BP} = \left\{\mathsf{inp}_1, \mathsf{inp}_2, \{B_{i,b_1,b_2}\}_{i\in[n],b_1,b_2\in\{0,1\}}\right\}$. *There exists a PPT simulator* $\mathsf{Sim}_{\mathsf{K}}$ *such that for every $x \in \{0,1\}^l$,*

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

We are now ready to describe $\mathsf{Sim}_{\mathsf{BP}}$.
$\mathsf{Sim}_{\mathsf{BP}}(1^n, 1^w, 1^\ell, P_x[1,w])$:

- If $P_x[1,w] \neq 0$, define the matrix A as $A := P_x[1,w] \cdot I_{w\times w}$. Else, $A :=$ "mirror-image" of $I_{w\times w}$.
- 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 as defined in Theorem 5.
- 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 now show that:

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

As a first step, we state the following lemma from Cramer et al. [23] that will be useful to prove the theorem.

Lemma 6. *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^TMy. In particular, there is a procedure* Extend, *running in time polynomial in n and w, that takes as input (x^TMy, x, y, M), where x, y and M are as defined in the above lemma and outputs X and Y such that XMY is $(x^TMy) \cdot I_{w\times w}$ if $x^TMy \neq 0$ else it is "mirror-image" of I.* [9]

[9] The "mirror-image" of a $w \times w$ identity matrix is also a $w \times w$ matrix such that the $(i, w-i+1)^{th}$ entry in the matrix is 1 and the rest of the entries in the matrix are 0.

We now proceed to proving that the output distributions of randBP and $\mathsf{Sim}_{\mathsf{BP}}$ are identical. We first define a sequence of hybrids such that the first hybrid is the real experiment (which is randBP) while the last hybrid is the simulated experiment (which is $\mathsf{Sim}_{\mathsf{BP}}$). Then, we show that the output distribution of each 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 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 describe Hybrid_1 as follows. The input to Hybrid_1 is $\widehat{\mathsf{BP}} = \mathsf{BP}\big|_x = (B_{i,x_{\mathsf{inp}_1(i)},x_{\mathsf{inp}_2(i)}})_{i\in[n]}$. 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 $n+1$ 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 Hybrid_0.

$\underline{\mathsf{Hybrid}_2}$: Hybrid_2 is same as Hybrid_1 except the way we compute $\tilde{s}$ and $\tilde{t}$. The input to 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 $n+1$ 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 Extend on input (c, e_1, e_w, P) to obtain $w \times w$ matrices S and T as described in Lemma 6. 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 $(\tilde{s}, \{\tilde{M}_i\}_{i\in[n]}, \tilde{t})$.

Hybrid_1 and Hybrid_2 differ only in the way $\tilde{s}$ and $\tilde{t}$ are computed. In 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 e_1 and hence, $\tilde{s} = e_1 \cdot R_0^{-1}$, which is same as the value in Hybrid_1. Similarly, we can show this for $\tilde{t}$.

$\underline{\mathsf{Hybrid}_3}$: This is same as the simulated experiment. That is, it takes as input $(1^n, 1^w, 1^\ell)$ and $P_x[1, w]$ and then executes $\mathsf{Sim}_{\mathsf{BP}}(1^n, 1^w, 1^\ell, P_x[1, w])$. The output of Hybrid_3 is the output of $\mathsf{Sim}_{\mathsf{BP}}$.

We now argue that Hybrid_2 and Hybrid_3 are identically distributed. First note that in Hybrid_2, $c = P[1, w]$. Then it follows from Lemma 6 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. Theorem 5 can be used to show that hybrids Hybrid_2 and Hybrid_3 are identically distributed. This shows that the output distribution of Hybrid_0 is identically distributed to Hybrid_3. This completes the proof. □

We now move to the second step where we show how to randomize the branching program using the procedure randBP' in such a way that the product of the matrices (which will be a 1×1 matrix) corresponding to an input only reveals the output of the function and nothing else. To achieve this, we need to ensure that the product of the matrices corresponding to one input is not correlated to the product of matrices corresponding to a different input, where both the inputs are such that they evaluate to 1. We solve this by multiplying the matrix B_{i,b_1,b_2} by α_{i,b_1,b_2} (which is picked at random). This ensures that multiplying the matrices corresponding to two different inputs result in two different products of α's which are mutually independent which in turn makes it feasible to achieve simulation of these matrices by just knowing the value of the function. We now describe the procedure randBP'. Note that randBP' takes as input $\widetilde{\mathsf{BP}}$ which is the output of randBP on the relaxed matrix branching program BP. s

Proceduce randBP'. In this procedure, we describe how to further randomize the output of randBP and then show how to simulate this by having just the output of BP. The input to randBP' is a 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 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 3. *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 α_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 $x \in \{0,1\}^\ell$.

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

- If $\mathsf{BP}(x) = 0$, output whatever $\mathsf{Sim}_{\mathsf{BP}}(1^s, 0)$ outputs. Else, pick a α 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 ℓ 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

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

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

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

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 randBP and 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 β_{V_1} denote $(\prod_i N_i)[1, w]$. Then by Theorem 4, $\Pr[\mathsf{randBP}(V_1) = V] = \Pr[\mathsf{Sim}_{\mathsf{BP}}(1^s, \beta_{V_1}) = V]$. 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 3, $\sum_{V_1 s.t. \beta_{V_1} = \alpha} \Pr[\mathsf{randBP}'(V_2) = V_1] = \frac{1}{p-1}$. 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 Ideal Graded Encoding Model

In this section, we describe the ideal graded encoding model. This section has been taken almost verbatim from [4]. 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 [26]. 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 5. *Given a ring R and a universe set $\mathbb{U}$, an element is a pair (α, S) where $\alpha \in R$ is the* value *of the element and $S \subseteq \mathbb{U}$ is the* index *of the element. Given an element e we denote by $\alpha(e)$ the value of the element, and we denote by $S(e)$ the index of the element. We also define the following binary operations over elements:*

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

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 do 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 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 [4]. 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 6. *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 4 (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 5 (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 dual-input oblivious relaxed matrix branching program BP of length n, width w, input length ℓ:

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

such that inp_1 and inp_2 are evaluation functions mapping $[n] \to [\ell]$, and each $B_{i,b_1,b_2} \in \{0, 1\}^{w\times w}$ is a full rank matrix.

We make a simplifying assumption that every input bit is inspected by BP exactly ℓ' number of times. We denote the set of indices that inspect the input bit j by $\mathsf{ind}(j)$.

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

Step 1: Randomizing the relaxed matrix branching program BP**.** The obfuscator $\mathcal{O}$ randomizes the branching program in two steps using procedures randBP and randBP' described in Section 4. It begins by sampling a large enough prime p of $\Omega(n)$ bits.

1. It invokes the procedure randBP on the relaxed matrix branching program BP obtained above to get $\widetilde{\mathsf{BP}} = \left(\mathsf{inp}_1,\ \mathsf{inp}_2,\ \tilde{s},\ \left\{\tilde{B}_{i,b_1,b_2}\right\}_{i\in[n],b_1,b_2\in\{0,1\}}, \tilde{t}\right)$. 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 then executes the procedure randBP' on input $\widetilde{\mathsf{BP}}$ to obtain $(\tilde{s},\ \{C_{i,b_1,b_2}\}_{i\in[n],b_1,b_2\in\{0,1\}},\ \tilde{t})$. The matrices C_{i,b_1,b_2} are such that $C_{i,b_1,b_2} = \alpha_{i,b_1,b_2} \cdot \tilde{B}_{i,b_1,b_2}$, where $\alpha_{i,b_1,b_2} \in \mathbb{Z}_p$ with $i \in [n], b_1, b_2 \in \{0,1\}$ are picked uniformly at random.

The output of this phase is $(\mathsf{inp}_1, \mathsf{inp}_2, \tilde{s},\ \{C_{i,b_1,b_2}\}_{\substack{i\in[n],\\ b_1,b_2\in\{0,1\}}},\ \tilde{t})$.

Looking ahead, the final obfuscation of BP will consist of ideal encodings of these elements with respect to a carefully chosen set system. Next, we describe how these sets are chosen.

Step 2: Initialization of the set systems. 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 $|\mathsf{ind}(j)| = \ell'$ sets in the system for each $j \in [\ell]$. We now associate the entries in the straddling set system $\mathbb{S}^j$ with the indices of BP which depend on x_j, i.e. the set $\mathsf{ind}(j)$. More precisely, let

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

Step 3: Associating elements of randomized RMBP with sets. Next, we associate a set to each element output by the randomization step. Recall that in a dual-input relaxed matrix branching program, each step depends on two fixed bits in the input defined by the evaluation functions inp_1 and 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:

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

Step 4: Encoding of elements in randomized RMBP. We 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 it asks for the encodings of the following elements:

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

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

$\mathcal{O}$ receives a list of handles for these elements from $\mathcal{M}$. Let $[\beta]_S$ denote the handle to (β, 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, which is then output by $\mathcal{O}$.

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

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

Correctness of Evaluation. We first assume that none of the calls to $\mathcal{M}$ fail and show that $\mathcal{O}(\mathsf{BP})$ on x outputs 1 iff $\mathsf{BP}(x) = 1$. We denote $b_1^i = x_{\mathsf{inp}_1(i)}$ and $b_2^i = x_{\mathsf{inp}_2(i)}$ in the following equation. From the description of the evaluation above, $\mathcal{O}(\mathsf{BP})$ outputs 0 on x if and only if

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

We conclude with the following theorem and corollary which summarize our results.

Theorem 7. *There is a virtual black box obfuscator $\mathcal{O}$ in the idealized model for all poly-sized RMBPs. For a family of input-oblivious RMBPs of length n and width w, the obfuscation requires n levels of multilinearity over a field of size $p = 2^{\Omega(n)}$, the obfuscated program consists of nw^2 encodings of field elements, and its evaluation involves $O(nw^2)$ multilinear operations.*

The proof of the above theorem follows along the lines of Barak et al. [4]. We provide the formal proof in Appendix D.

The following corollary follows from Theorem 1, Theorem 2 and the above theorem.

Corollary 1. *There is a virtual black box obfuscator $\mathcal{O}$ in the idealized model for non-deterministic branching programs. For a family of keyed branching programs (or formulas) of size s, the obfuscation requires s levels of multilinearity over a field of size $p = 2^{\Omega(s)}$, the obfuscated program consists of $O(s^3)$ encodings of field elements, and its evaluation involves $O(s^3)$ multilinear operations. For a family of input-oblivious, special layered branching programs of length n and width w, the obfuscation requires n levels of multilinearity over a field of size $p = 2^{\Omega(n)}$, the obfuscated program consists of $O(nw^2)$ encodings of field elements, and its evaluation involves $O(nw^2)$ multilinear operations.*

In the above theorem and its corollary, the obliviousness requirement can be relaxed by incurring an additional multiplicative overhead of ℓ to the levels of multilinearity and the number of multilinear operations, where ℓ is the number of input variables.

Claim 6. *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 13) 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 1, 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 13. More formally,

Claim 7. *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^{bad} of e such that $D(e^{\mathsf{bad}})$ contains an invalid input-profile but decomposition of all sub-elements of e^{bad} have valid input-profiles. We now do a case analysis on the structure of e^{bad}.

e^{bad} cannot be a basic sub-element since input-profile of all basic sub-elements is valid. Also, e^{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^{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 4, 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 13, 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^{bad} is a sub-element of e, using Claim 13, 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 4 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 8. *$D(e)$ 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 5, 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 16, $\mathsf{Prof}(s_{11})$ 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 9. *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 $\mathsf{inp}_2(i) = k$, $\mathsf{Prof}(s)_k \neq *$. □

□

7.1 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 α's which is different for every in its decomposition. And hence, with negligible probability it happens that the α'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 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. 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 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 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 F to obtain the following distribution represented by the random variable $\mathcal{V}^{\mathsf{Sim}}_s$.
   $$\left\{\tilde{s}, \tilde{B}_{i,b^i_1,b^i_2}, \tilde{t} : i \in [n], b^i_1 = x_{\mathsf{inp}_1(i)}, b^i_2 = x_{\mathsf{inp}_2(i)}\right\}$$
   (c) We evaluate the polynomial p_s, which is the polynomial computed by the circuit $\alpha(s)$, on $\mathcal{V}^{\mathsf{Sim}}_s$. If the evaluation yields a non-zero result then Sim outputs "Non-zero".
3. For all $s \in D(e)$, if $p_s(\mathcal{V}^{\mathsf{Sim}}_s) = 0$ then 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 15 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 15 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 10. *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 16, 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 11. *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.

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

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 8. *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 18, 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.

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

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 18 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 2. *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 12. *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 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 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 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. We thank Jean-Sebastien Coron and Shai Halevi for helpful discussions regarding secure parameter sizes for multilinear map instantiations. We thank Ilan Komargodski and Alexander Sherstov for bringing to our attention the best results on optimizing depth of formulas. We also thank Omkant Pandey, Manoj Prabhakaran, and Mark Zhandry for several helpful comments. We are especially grateful to Ilan Komargodski for bringing the work of [31] to our attention as well as for several other helpful suggestions and comments. Finally, we thank the CCS Program Committee, and especially Mariana Raykova, for very useful feedback and suggestions.

References

[1] P. Ananth, D. Boneh, S. Garg, A. Sahai, and M. Zhandry. Differing-inputs obfuscation and applications. Cryptology ePrint Archive, Report 2013/689, 2013.

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

[3] B. Barak, N. Bitansky, R. Canetti, Y. T. Kalai, O. Paneth, and A. Sahai. Obfuscation for evasive functions, 2014.

[4] B. Barak, S. Garg, Y. T. Kalai, O. Paneth, and A. Sahai. Protecting obfuscation against algebraic attacks. In *EUROCRYPT*, 2014.

[5] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. P. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. In *CRYPTO*, pages 1–18, 2001.

[6] D. A. M. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC1. In *STOC*, 1986.

[7] M. Bellare and V. T. Hoang. Adaptive witness encryption and asymmetric password-based cryptography. Cryptology ePrint Archive, Report 2013/704, 2013. `http://eprint.iacr.org/`.

[8] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[9] M. Bellare, I. Stepanovs, and S. Tessaro. Poly-many hardcore bits for any one-way function. Cryptology ePrint Archive, Report 2013/873, 2013. `http://eprint.iacr.org/`.

[10] N. Bitansky, R. Canetti, O. Paneth, and A. Rosen. More on the impossibility of virtual-black-box obfuscation with auxiliary input. Cryptology ePrint Archive, Report 2013/701, 2013.

[11] M. L. Bonet and S. R. Buss. Size-depth tradeoffs for boolean formulae. *Information Processing Letters*, 49(3):151–155, 1994.

[12] E. Boyle, K.-M. Chung, and R. Pass. On extractability obfuscation. Cryptology ePrint Archive, Report 2013/650, 2013.

[13] E. Boyle and R. Pass. Limits of extractability assumptions with distributional auxiliary input. Cryptology ePrint Archive, Report 2013/703, 2013.

[14] Z. Brakerski and G. 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.

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

[16] R. P. Brent. The parallel evaluation of general arithmetic expressions. *Journal of the ACM (JACM)*, pages 201–206, 1974.

[17] C. Brzuska, P. Farshim, and A. Mittelbach. Indistinguishability obfuscation and UCEs: The case of computationally unpredictable sources. Cryptology ePrint Archive, Report 2014/099, 2014.

[18] N. H. Bshouty, R. Cleve, and W. Eberly. Size-depth tradeoffs for algebraic formulae. In *FOCS*, pages 334–341, 1991.

[19] R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004.

[20] R. Cleve. Towards optimal simulations of formulas by bounded-width programs. In *STOC*, pages 271–277, 1990.

[21] H. Cohn, S. Goldwasser, and Y. T. Kalai. The impossibility of obfuscation with a universal simulator. *CoRR*, abs/1401.0348, 2014.

[22] J.-S. Coron, T. Lepoint, and M. Tibouchi. Practical multilinear maps over the integers. In *CRYPTO*, pages 476–493, 2013.

[23] R. Cramer, S. Fehr, Y. Ishai, and E. Kushilevitz. Efficient multi-party computation over rings. In *EUROCRYPT*, pages 596–613, 2003.

[24] W. Diffie and M. E. Hellman. Multiuser cryptographic techniques. In *AFIPS National Computer Conference*, pages 109–112, 1976.

[25] U. Feige, J. Kilian, and M. Naor. A minimal model for secure computation. In *STOC*, pages 554–563, 1994.

[26] S. Garg, C. Gentry, and S. Halevi. Candidate multilinear maps from ideal lattices. In *EUROCRYPT*, pages 1–17, 2013.

[27] S. Garg, C. Gentry, S. Halevi, and M. Raykova. Two-round secure mpc from indistinguishability obfuscation. 2014.

[28] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, pages 40–49, 2013.

[29] S. Garg, C. Gentry, S. Halevi, and D. Wichs. On the implausibility of differing-inputs obfuscation and extractable witness encryption with auxiliary input. Cryptology ePrint Archive, Report 2013/860, 2013.

[30] C. Gentry, S. Halevi, M. Raykova, and D. Wichs. Outsourcing private ram computation. Cryptology ePrint Archive, Report 2014/148, 2014.

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

[32] S. Goldwasser, V. Gordon, Dov Goyal, A. Jain, J. Katz, F.-H. Liu, E. Shi, H.-S. Zhou, and A. Sahai. Multi-input functional encryption. 2014.

[33] V. Goyal, Y. Ishai, A. Sahai, R. Venkatesan, and A. Wadia. Founding cryptography on tamper-proof hardware tokens. In *TCC*, pages 308–326, 2010.

[34] V. Goyal, A. Jain, V. Koppula, and A. Sahai. Functional encryption for randomized functionalities. Cryptology ePrint Archive, Report 2013/729, 2013.

[35] S. Hohenberger, A. Sahai, and B. Waters. Replacing a random oracle: Full domain hash from indistinguishability obfuscation. 2014.

[36] S. Jukna. *Boolean Function Complexity - Advances and Frontiers*, volume 27 of *Algorithms and combinatorics*. Springer, 2012.

[37] V. Khrapchenko. On a relation between the complexity and the depth. *Metody Diskretnogo Analiza Synthesis of Control Systems*, 32:76–94, 1978.

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

[39] I. Komargodski, M. Naor, and E. Yogev. Secret-sharing for np from indistinguishability obfuscation. Cryptology ePrint Archive, Report 2014/213, 2014.

[40] A. Langlois, D. Stehlé, and R. Steinfeld. Gghlite: More efficient multilinear maps from ideal lattices. In *EUROCRYPT*, pages 239–256, 2014.

[41] H. T. Lee and J. H. Seo. Security analysis of multilinear maps over the integers. In *CRYPTO (1)*, pages 224–240, 2014.

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

[43] W. J. Masek. A fast algorithm for the string editing problem and decision graph complexity, 1976.

[44] T. Moran and A. Rosen. There is no indistinguishability obfuscation in pessiland. Cryptology ePrint Archive, Report 2013/643, 2013.

[45] O. Pandey, M. Prabhakaran, and A. Sahai. Obfuscation-based non-black-box simulation and four message concurrent zero knowledge for np. Cryptology ePrint Archive, Report 2013/754, 2013.

[46] R. Pass, K. Seth, and S. Telang. Indistinguishability obfuscation from semantically-secure multilinear encodings. In *CRYPTO*, pages 500–517, 2014.

[47] M. Paterson and U. Zwick. Shallow circuits and concise formulae for multiple addition and multiplication. *Computational Complexity*, 3:262–291, 1993.

[48] F. P. Preparata and D. E. Muller. Efficient parallel evaluation of boolean expression. *IEEE Trans. Computers*, 25(5):548–549, 1976.

[49] A. Sahai and B. Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *STOC*, pages 475–484, 2014.

[50] M. Sauerhoff, I. Wegener, and R. Werchner. Relating branching program size and formula size over the full binary basis. In *STACS*, pages 57–67, 1999.

[51] I. S. Sergeev. Upper bounds for the formula size of symmetric boolean functions. *Russian Mathematics*, 58 (5):30–42, 2014.

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

A Boolean Formulae

A boolean circuit for a function $f : \{0,1\}^\ell \to \{0,1\}$ is a directed acyclic graph (DAG). The vertices in this graph are either input variables or gates. We assume that all the gates in the circuit have fan-in at most 2. 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 the out-degree of 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. A formula can be viewed as binary tree where the root is the output gate. We define the *size* of a formula to be the number of leaves in this binary tree.

B From Formula to BP

In this section, we give a transformation of boolean formulas over AND and NOT gates to a branching program. Note that any formula over AND, OR and NOT gates can be converted to a formula over AND and NOT gates of the same size.
Consider a formula, denoted by F. We inductively transform F to a branching program BP. Our construction will maintain a stronger induction hypothesis. There will be two sink nodes, "accept" and "reject." Also, there will be a path from the source to the accept iff the output is 1 and there will be a path from the source to the reject iff the output is 0.

The base case corresponds to an input wire w. Let input variable be x_i. We construct a branching program for w, denoted by BP_w consists of three nodes denoted by source, acc and rej. We add an edge labeled 0 from source to rej and an edge labeled 1 from source to acc. We label the source with x_i.
We proceed to the induction hypothesis. Consider a gate G.
Case (1) and gate:- Let F_1 and F_2 be two sub-formulae such that their output wires are fed to F. Let $\mathsf{BP}_{\mathsf{F}_1}$ and $\mathsf{BP}_{\mathsf{F}_2}$ be the branching programs for F_1 and F_2, respectively. We construct a branching program for F as follows (see Figure 1). We merge the accept node of $\mathsf{BP}_{\mathsf{F}_1}$ with the source node of $\mathsf{BP}_{\mathsf{F}_2}$. Similarly, merge the reject node of $\mathsf{BP}_{\mathsf{F}_1}$ with the reject node of $\mathsf{BP}_{\mathsf{F}_2}$.
Case (2) not gate:- Let F' be the sub-formula such that the output wire of F' is fed into the gate G. Let $\mathsf{BP}_{\mathsf{F}'}$ be a branching program for F'. To construct the branching program for F we

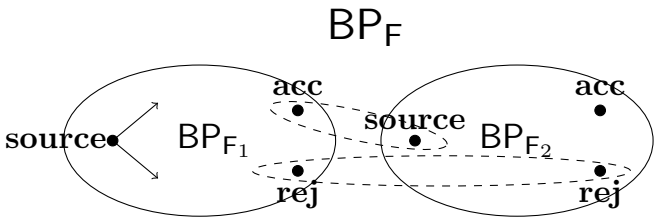

Figure 1: This denotes the branching program for an AND gate.

simply rename accept node of $\mathsf{BP}_{\mathsf{F}'}$ as reject node for BP_F. We also rename reject node of $\mathsf{BP}_{\mathsf{F}'}$ as accept node for BP_F.

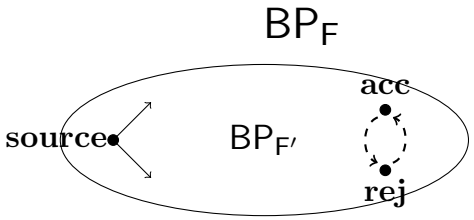

Figure 2: This denotes the branching program for a NOT gate. The accept and the reject nodes are interchanged.

Note that once the transformation is complete for the formula, the final reject node can be deleted. So our final construction will only have one sink node, the "accept" node.

It is easy to see that the above described layered branching program correctly evaluates the formula F. More formally,

Lemma 7. *For every input $x \in \{0,1\}^l$, we have $\mathsf{F}(x) = 1$ if and only if $\mathsf{BP}_\mathsf{F}(x) = 1$. Moreover, for a formula of size s, the size of the branching program BP_F is at most s.*

Proof Sketch. It follows by an induction on the structure of the formula by noting that the number of leaves in a formula is the sum of the leaves of the left sub-tree and the right sub-tree.

C Proof of Theorem 3

At a high level the proof proceeds as follows. We demonstrate a transformation that consists of the following steps – first the formula is balanced and then the resulting balanced formula is converted to a linear bijection straightline-program (LBSP) which is then converted to an RMBP. Note that the size of the resulting relaxed matrix branching program is better than the one presented above. For completeness sake, we present the definition of LBSP as given in [20].

Definition 7. *[20] A linear bijection straightline-program (LBSP) over $\mathbb{Z}_p$ is a sequence of assignments of the form:*

$$R_j \leftarrow R_j + c \cdot R_i$$

$$R_j \leftarrow R_j - c \cdot R_i$$

$$R_j \leftarrow R_j + x_u \cdot R_i$$

$$R_j \leftarrow R_j - x_u \cdot R_i$$

where $R_1, \ldots, R_w$ are registers, $x_1, \ldots, x_l$ are inputs, $c \in \mathbb{Z}_p$ and $u \in \{1, \ldots, l\}$. The width of LBSP, denoted by w, is the number of the registers in LBSP. The length of a LBSP is the number of statements it contains. An LBSP computes a function $f(x_1, \ldots, x_l)$ if there exists $a, b \in \{1, \ldots, w\}$ such that the register R_a, after the evaluation, contains $R_a^* + R_b^* \cdot f(x_1, \ldots, x_l)$, where R_a^* denotes the initial value of the register R_a and R_b^* contains the initial value of the register R_b.

In the LBSP we are going to consider, we will assume that the initial value in the register R_b is 1 and the initial value in the register R_a is 0.

We now provide a sketch of the proof of Theorem 3. [10] We refer the reader to Giel [31] for more details. The main deviation of our proof from the proof by [31] is in the last step – instead of transforming a LBSP to a layered branching program, we directly transform LBSP to a relaxed matrix branching program.

Proof sketch. Consider a formula of size s over any complete basis. First, the formula is balanced using [11] to obtain a formula of depth at most $3c \ln(2) \log s$ and the size of the formula is at most s^α, where $c \geq 2$ is a constant and $\alpha = 1 + \big(1/(1 + \log(c-1))\big)$. Then the balanced formula is converted to a linear bijection straight-line program (LBSP) using [31]. The resulting LBSP has width at most $2^k + 2$ and the number of variable references (which is essentially the number of instructions in LBSP, each variable appears in) is at most $8s^{\alpha + \frac{6c \ \ln 2}{k}}$, where k is a constant to be determined later. Finally, we transform this as a relaxed matrix branching program, denoted by $\mathsf{BP} = \{B_{i,b}\}_{i \in [n+1], b \in \{0,1\}}$, each matrix of width $w \times w$, as follows. Note that matrices $B_{i,b}$ represent the $(i-1)^{th}$ instruction in LBSP. Further, we associate a function $\mathsf{inp} : [n] \to [\ell]$, where n is the number of matrices that we obtain.

- $k = 1$: In this case, both $B_{1,0}$ and $B_{1,1}$ represent the same matrix. The first row of the matrix $B_{1,0}$ represents the initial values in the registers in LBSP. The rest of the rows are picked such that $B_{1,0}$ is a full rank matrix. The function inp on k is set to 1.
- $k > 1$ and the $(k-1)^{th}$ instruction is of the form $R_j \leftarrow R_j + c \cdot R_i$ OR $R_j \leftarrow R_j - c \cdot R_i$ for $c \in \mathbb{Z}_2$: Even in this case both $B_{i,0}$ and $B_{i,1}$ represent the same matrix. $B_{i,0}$ contains 1's on the diagonal. It also contains 1 in the $(i,j)^{th}$ entry. The rest of the entries are 0. Even in this case, inp on k is set to 1.
- $k > 1$ and the $(k-1)^{th}$ instruction is of the form $R_j \leftarrow R_j + x_u \cdot R_i$ OR $R_j \leftarrow R_j - x_u \cdot R_i$ for $u \in \{1, \ldots, n\}$: In this case, $B_{1,0}$ is an identity matrix of width w. The matrix $B_{i,1}$ contains 1's on the diagonal and it contains 1 in the $(i,j)^{th}$ entry. The rest of the entries in the matrix are 0. For this case, $\mathsf{inp}(k)$ is set to u.

This completes the description of RMBP. Define $M_i = B_{i, x_{\mathsf{inp}(i)}}$, for some $x \in \{0,1\}^\ell$. Now, observe that the product $\prod_{i=1}^{j+1} M_i$ denotes the evaluation of the first i instructions in LBSP. Finally, the $(1,a)^{th}$ entry in the final product $\prod_{i=1}^{n+1} M_i$ denotes the output of the LBSP on input x. Further, observe that the width of the relaxed matrix branching program is exactly the width of the LBSP which is at most $2^k + 2$ and the length of the LBSP is at most $8s^{\alpha + \frac{6c \ \ln 2}{k}}$. By suitably substituting for k, we can get the size of the RMBP to be $s^{1+\epsilon}$, for any $\epsilon > 0$ and the width of the matrices in this RMBP is a constant (depending on ϵ).

D 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.

[10] In the proof of Giel's theorem, the ring $\mathbb{Z}_2$ was used in the definition of LSBP. We remark that the proof extends to the case when the LSBP is considered in the ring over $\mathbb{Z}_p$.

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 **??** that for any formula F there exists a RMBP represented in the form of $O(|\mathsf{F}|)$ matrices each of width $O(|\mathsf{F}|)$. 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 $n(|\mathsf{F}|)$ and width $w(|\mathsf{F}|)$ computing on $\ell(|\mathsf{F}|)$ 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 Sim Here we construct a simulator Sim that takes as input $1^{|\mathsf{F}|}$ 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 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, Sim initializes the oracle $\mathcal{M}$ with formal variables. Also note that the simulator can compute the evaluation functions inp_1 and inp_2 and also the system used for encodings since the RMBPs are oblivious. This would be important when 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, Sim will emulate the oracle $\mathcal{M}$ that $\mathcal{O}$ accesses as follows: Sim will maintain a table of handles and corresponding level of encodings that have been initialized so far. As mentioned before, Sim will initialize the oracle $\mathcal{M}$ with formal variables. Note that 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 Sim completes the emulation of $\mathcal{O}$ it obtains a simulated obfuscation $\tilde{\mathcal{O}}(\mathsf{F})$. Now Sim has to simulate the view of the adversary on input $\tilde{\mathcal{O}}(\mathsf{F})$. Our Sim will use the same handles table for emulating the oracle calls of both $\mathcal{O}$ and $\mathcal{A}$. Hence, 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 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, 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 α_{i,b_1,b_2}. Using this we prove that value of each single input element depends on product of different α'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.

D.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 Sim to decompose elements into single-input elements. Parts of this description have been taken verbatim from [4]. 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 1. *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 15). We begin by proving a claim about the relation between the level of encoding of e and a sub-element e' of e.

Claim 13. *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 13) 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 1, 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 13. More formally,

Claim 14. *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^{bad} of e such that $D(e^{\mathsf{bad}})$ contains an invalid input-profile but decomposition of all sub-elements of e^{bad} have valid input-profiles. We now do a case analysis on the structure of e^{bad}.

e^{bad} cannot be a basic sub-element since input-profile of all basic sub-elements is valid. Also, e^{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^{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 4, 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 13, 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^{bad} is a sub-element of e, using Claim 13, 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 4 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 15. *$D(e)$ 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 5, 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 16, $\mathsf{Prof}(s_{11})$ 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 16. *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 $\mathsf{inp}_2(i) = k$, $\mathsf{Prof}(s)_k \neq *$. □

□

D.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 α's which is different for every in its decomposition. And hence, with negligible probability it happens that the α'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 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. 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 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 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 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 Sim outputs "Non-zero".
3. For all $s \in D(e)$, if $p_s(\mathcal{V}_s^{\mathsf{Sim}}) = 0$ then 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 15 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 15 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 17. *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 16, 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 18. *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.

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

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:

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

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 18, we have that $\Pr_{\mathcal{V}_C^{\mathsf{real}}}[p_e(\mathcal{V}_C^{\mathsf{real}}) = 0] = 0$ with negligible probability. Further since $p_e = \sum_{s \in D(e)} p_s$, we have the following.

$$\begin{aligned} \Pr_{\mathcal{V}_C^{\mathsf{real}}}[p_e(\mathcal{V}_C^{\mathsf{real}}) = 0] &= \Pr_{\mathcal{V}_C^{\mathsf{real}}}[\exists s \in D(e) \ : \ p_s(\mathcal{V}_C^{\mathsf{real}}) = 0] \\ &= \mathsf{negl}(n) \end{aligned}$$

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 18 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 19. *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 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 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 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. □