How to Obfuscate Programs Directly

Joe Zimmerman*

Abstract

We propose a new way to obfuscate programs, using composite-order multilinear maps. Our construction operates directly on straight-line programs (arithmetic circuits), rather than converting them to matrix branching programs as in other known approaches. This yields considerable efficiency improvements. For an NC^1 circuit of size s and depth d, with n inputs, we require only $O(d^2s^2 + n^2)$ multilinear map operations to evaluate the obfuscated circuit—as compared with other known approaches, for which the number of operations is exponential in d. We prove virtual black-box (VBB) security for our construction in a generic model of multilinear maps of hidden composite order, extending previous models for the prime-order setting.

Our scheme works either with "noisy" multilinear maps, which can only evaluate expressions of degree λ^c for pre-specified constant c; or with "clean" multilinear maps, which can evaluate arbitrary expressions. The "noisy" variant can be instantiated at present with the Coron-Lepoint-Tibouchi scheme, while the existence of "clean" maps is still unknown. With known "noisy" maps, our new obfuscator applies only to NC^1 circuits, requiring the additional assumption of FHE in order to bootstrap to P/poly (as in other obfuscation constructions).

From "clean" multilinear maps, on the other hand (whose existence is still open), we present the first approach that would achieve obfuscation for P/poly directly, without FHE. We also introduce the concept of *succinct* obfuscation, in which the obfuscation overhead size depends only on the length of the input and of the *secret* part of the circuit. Using our new techniques, along with the assumption that factoring is hard on average, we show that "clean" multilinear maps imply *succinct* obfuscation for P/poly. For the first time, the only remaining obstacle to implementable obfuscation in practice is the noise growth in known, "noisy" multilinear maps. Our results demonstrate that the question of "clean" multilinear maps is not a technicality, but a central open problem.

*Stanford University, jzim@cs.stanford.edu.

1 Introduction

Program obfuscation is the task of making code "unintelligible", so that the obfuscated code reveals nothing about the implementation beyond its functionality. Obfuscation has many practical applications, such as intellectual property protection and software watermarking, as well as applications to basic cryptographic primitives [DH76, BGI+01].

The theoretical study of obfuscation was initiated by Barak, Goldreich, Impagliazzo, Rudich, Sahai, Vadhan, and Yang [BGI+01]. In that work, the authors also showed that general-purpose program obfuscation could not achieve the natural definition of virtual black-box secuity (VBB), which led many to suspect that a useful general-purpose obfuscator was impossible. This view changed with the work of Garg, Gentry, Halevi, Raykova, Sahai, and Waters [GGH+13b], who proposed a general-purpose obfuscator based on the powerful primitive of *multilinear maps* [BS03], as constructed by Garg, Gentry, and Halevi [GGH13a], Coron, Lepoint, and Tibouchi [CLT13], and Gentry, Gorbunov, and Halevi [GGH14].

For their general-purpose obfuscator, Garg et al. [GGH+13b] proved the weaker notion of indistinguishability obfuscation ($i\mathcal{O}$) [BGI+01] in a generic model of encoded matrices. Subsequently it has been shown that in a generic model of multilinear maps, general-purpose obfuscation can even achieve VBB security [BR14, BGK+14], and that $i\mathcal{O}$ can be based on a single, instance-independent security assumption [GLSW14]. Sahai and Waters have also shown that even the weaker notion of $i\mathcal{O}$ has many cryptographic applications, via the technique of "punctured programs" [SW14]. Since then, obfuscation has become an extremely active area of study, and many other applications and complexity-theoretic implications have been explored; see [AGIS14] for an overview.

Even with known constructions and applications, however, general-purpose obfuscation is currently not feasible to implement in practice. The work of Ananth, Gupta, Ishai and Sahai [AGIS14] investigates the question of optimizing obfuscation, and obtains significant improvements for the specific case of Boolean formulas, but much work remains to be done. One major source of inefficiency is that in all known constructions, including that of Ananth et al. [AGIS14], obfuscation requires converting the input circuit to a matrix branching program, which incurs a considerable cost in performance.

1.1 Our Results

In this work, we propose a new way to construct obfuscation, which operates directly on straight-line programs (arithmetic circuits, Section 2.3), without converting them to matrix branching programs. The evaluation of an obfuscated circuit mirrors the structure of the original circuit.

Our construction is based on asymmetric composite-order multilinear maps [BS03, GGH13a, CLT13, GGH14]. It can operate either with "noisy" multilinear maps (which we know how to instantiate, via the CLT scheme), or with "clean" maps, whose existence is still open. In the case of "noisy" multilinear maps, our construction (like others) is limited to NC^1, and requires FHE to bootstrap to P/poly. With "clean" multilinear maps, on the other hand, we show that we would be able to obfuscate P/poly directly, without the prohibitively expensive bootstrapping step via FHE. Indeed, if we knew how to construct "clean" multilinear maps, then our results in this work would *immediately* yield obfuscation for P/poly, with parameters that could be feasible in practice.

In addition to qualitatively new results, our techniques yield considerable performance improvements even for existing, "noisy" multilinear maps. For instance, for circuits of size s and depth d with n inputs, we require only $O(d^2s^2+n^2)$ multilinear map elements and operations (Table 1). All other known approaches require a number exponential in the circuit's depth, since every sub-circuit with fanout > 1 must be duplicated before converting the circuit to a matrix branching program.

Perspective: towards implementable obfuscation. Currently, general-purpose obfuscation is not feasible to implement in practice. There have been two main obstacles to its implementation. The first is that, in known ("noisy") multilinear maps such as the GGH and CLT schemes, the noise—and hence the parameters—grow with the *degree* of the polynomial being computed over encoded elements; this limits us to NC^1 circuits, because the degree of a circuit may increase exponentially with its depth. The second obstacle is that, prior to this work, obfuscation required converting the input circuit to a matrix branching program, whose size in general is also exponential in the depth of the original circuit.

This work removes the second obstacle. In our construction, the number of multilinear map operations is polynomial in the circuit size; it is only the degree of multilinearity (and hence the noise growth in "noisy" multilinear maps) that restricts our construction to NC^1. If we could construct "clean" multilinear maps, then our results would immediately yield obfuscation for P/poly, with parameters that could be feasible in practice. In our view, our results indicate that constructing "clean" multilinear maps is one of the most fundamental open problems in cryptography.

Succinctness and keyed circuits. Our new approach is particularly effective for obfuscating *keyed* circuit families $(C(\cdot, \mathbf{y}))_{\mathbf{y}\in\{0,1\}^m}$ (Section 2.4), in which the circuit's structure C is public, and one only needs to hide a short secret key $\mathbf{y} \in \{0,1\}^m$ embedded in the circuit—as is common in many cryptographic applications. For example, for a keyed circuit $C : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$ of size s and depth d (with n inputs and key length m), our obfuscation consists of only $O(m+n^2)$ ring elements in the multilinear map, and evaluation requires $O(s + n^2)$ ring operations, with multilinearity degree $O(2^d + n^2)$ (Section 4.3.1).

For keyed circuits, we also define *succinct* obfuscation (Section 2.10), in which the obfuscation overhead size depends only on the input length n and the secret key length m, and is independent of the circuit size. Using our new techniques, along with the assumption that factoring is hard on average, we show that "clean" multilinear maps would imply *succinct* obfuscation for all of P/poly.

Of course, we can regard every circuit family as keyed, by viewing the original circuit as the secret key input to the universal circuit. In this case, succinctness means that the obfuscation overhead size depends only on the size of the part of the original circuit that the obfuscation needs to hide (as well as on the input length). However, the keyed model is especially natural, and we expect that in most applications it will find more use than general-purpose obfuscation.

New design spaces. When the obfuscator converts every circuit C to a matrix branching program, as in previously known approaches, it usually does not help to optimize the design of C itself. The depth of C determines the size of the resulting branching program,[1] but apart from that, every design strategy results in the same procedure to evaluate the obfuscated circuit $\mathcal{O}(C)$, and the same performance—namely, a series of matrix multiplications of encoded elements in the multilinear map.

By contrast, with our new techniques, the obfuscated program's evaluation mirrors the structure of the original arithmetic circuit. If these circuits are naturally keyed, as in most cryptographic applications, then the performance changes considerably with the design strategy, and we expose a rich new design space. The execution of any machine—say, a Turing machine or RAM—can be converted to a circuit with overhead at most polylogarithmic,[2] as long as the machine is already *oblivious* (Section 2.2)—i.e., its control flow does not depend on its input data. This means that any tools for designing efficient oblivious algorithms now apply to program obfuscation.

[1] In some cases, for Boolean formulas, the size of the branching program may depend on the formula's size [AGIS14].

[2] For instance, in some models there is overhead involved in decomposing word operations into bits.

	Degree of multilinearity	Obfuscation size (# ring elements)	Evaluation time (# ring operations)
Via Barrington's Thm. [GGH+13b, BR14, BGK+14]	$O(4^d n + n^2)$	$O(4^d n + n^2)$	$O(4^d n + n^2)$
[AGIS14]	$O(2^d n + n^2)$	$O(8^d n + n^2)$	$O(8^d n + n^2)$
[AGIS14] + [Gie01]	$O(2^{(1+\varepsilon)d} n + n^2)$	$O(2^{(1+\varepsilon)d} 4^{2/\varepsilon} n + n^2)$	$O(2^{(1+\varepsilon)d} 4^{2/\varepsilon} n + n^2)$
This work	$O(2^d n + n^2)$	$\boldsymbol{O(d^2 s^2 + n^2)}$	$\boldsymbol{O(d^2 s^2 + n^2)}$

Table 1: Performance for circuits of input length n, size s, and depth d. We always have $n, s < O(2^d)$, since the gates have fanin two; and in most applications we have $n, s \ll 2^d$. For moderately "narrow" circuits with $s < O(dn)$ and $d > 2 \lg n$, for example, we have $O(d^2 s^2 + n^2) = O(d^4 n^2) = o(2^d n)$. We present the cost here in terms of ring elements and ring operations. The concrete cost in bits and bit operations depends on the performance of the multilinear map (Section 2.8); for "clean" maps (whose existence is still open), the cost is just $\text{poly}(\lambda)$, while for the CLT scheme [CLT13], the reader should multiply every obfuscation size and evaluation time by $O(\deg^2) \cdot \text{poly}(\lambda)$, where deg is the corresponding multilinearity degree from the first column.

For example, to specialize our new construction to Boolean formulas, we use an efficient oblivious stack [HS66, PF79, MZ14] to evaluate the formulas in postfix order, and we rely on the Fast Fourier Transform (FFT) to reduce the degree of the resulting computation (Section 4.3.3). We believe that these applications are only the beginning, and we hope that this work will encourage further study of obfuscating *specific*, keyed circuit families. This goal is closely related to the design of efficient oblivious algorithms for specific problems, which is of independent interest in secure multi-party computation and other areas of cryptography. More broadly, while the existence of general-purpose obfuscation is an important theoretical result, we believe that its role in applications is actually quite limited; it is analogous to running all of our programs on a universal Turing machine.

VBB security in the generic model. Since obfuscation is such a powerful primitive, historically it has been difficult to prove constructions secure based on simple, falsifiable assumptions. In the first candidate construction [GGH+13b], Garg et al. prove indistinguishability obfuscation ($i\mathcal{O}$) based on a meta-assumption which roughly asserts that the scheme is secure, which they validate in a generic model of generic (encoded) matrices. Brakerski and Rothblum [BR14] and Barak et al. [BGK+14] develop these results further, showing how to extend the obfuscation paradigm of [GGH+13b] to achieve the much stronger definition of *virtual black-box* (VBB) security in a very natural generic model of multilinear maps, similar to the generic group model [Sho97]. In this work, we also prove VBB security, in a generic model similar to that of [BR14, BGK+14], adapted to the setting of (hidden) composite order.

As observed by Brakerski and Rothblum [BR14], it is not clear how we should interpret a proof of VBB in a generic model, since we know that VBB security in the standard model is impossible for general circuit families [BGI+01]. However, as far as we know, it may be possible to achieve VBB obfuscation for many specific classes of circuits, even if not for the pathological examples in the negative results of [BGI+01]. We also do not know any (unconditional) negative results for $i\mathcal{O}$, and a proof of VBB in the generic model also implies $i\mathcal{O}$ in the generic model. Thus, it is plausible that our construction achieves $i\mathcal{O}$ for all circuits (or some intermediate definition, such as differing-inputs obfuscation [BGI+01, ABG+13, BCP14]), and a generic-model VBB proof serves as evidence of this as well.

More generally, a generic-model VBB proof shows that a scheme resists a wide class of "algebraic" attacks, and that any attack that breaks VBB security must exploit some property of the concrete instantiation of multilinear maps.[3] As in the random oracle model, we know that no real primitive can actually instantiate the generic model in all cases [CGH98], and we view the negative result for VBB as another example of that paradigm. In this work, as in other works that rely on generic models [GGH+13b, BR14, BGK+14], we believe that a generic-model proof provides strong heuristic evidence that the corresponding (meta-)assumptions usually hold in the standard model. Of course, it would be even better to prove our construction secure based on a single (instance-independent) falsifiable assumption, as in the work of Gentry et al. [GLW14, GLSW14, GGHZ14]. We leave this as an important open problem for future work.

Extensions. We observe that our techniques can be naturally extended to *functional encryption* [O'N10, BSW11] (as well as its generalization, multi-input functional encryption [GGG+14]), enabling direct constructions that do not require the full machinery of obfuscation and NIZK proofs, and hence avoid their considerable performance cost. We now outline one approach to this extension; we defer the full details to an upcoming work. First we note that in our obfuscation construction, we give out an obfuscated *keyed* circuit, $\mathcal{O}(C(\cdot, \mathbf{y}))$, which acts much like the functional decryption key $f_{C(\cdot,\mathbf{y})}$ in a functional encryption scheme. The evaluator can select arbitrary inputs $\mathbf{x} \in \{0,1\}^n$ of her choice, and use the obfuscated circuit to learn $C(\mathbf{x}, \mathbf{y})$. In functional encryption, however, the evaluator has an additional ability: she can "defer" the evaluation of $C(\mathbf{x}, \mathbf{y})$, by running $\mathsf{ct}_{\mathbf{x}} \leftarrow \mathrm{Enc}(\mathsf{pk}, \mathbf{x})$; then, roughly speaking, an adversary who obtains the value $\mathsf{ct}_{\mathbf{x}}$ learns nothing about $\mathbf{x}$, except those outputs $C(\mathbf{x}, \mathbf{y})$ for which the adversary has the corresponding keys $f_{C(\cdot,\mathbf{y})}$. So, to generalize our obfuscation construction to functional encryption, we need to enable the evaluator to "defer" an input $\mathbf{x}$ in this fashion. Since our construction already represents each input bit $x_1, \ldots, x_n \in \mathbf{x}$ as an encoded element in the multilinear map, this amounts to generating $\mathrm{poly}(\lambda)$ additional encoded elements, of which we can use a subset to "blind" an encoded input $\mathbf{x}$, constructing the ciphertext for the functional encryption scheme.

Another natural extension of our construction is to obfuscate circuits with *multi-bit output*,[4] $C : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^\ell$ for $\ell > 1$. We describe the details of this extension below, in Remark 3.18. Intuitively, since our evaluation of an obfuscated circuit follows the structure of the original circuit, we can also reuse intermediate results for gates with fanout > 1, and we need not repeat the entire computation for each bit of the output (as we would in approaches based on Barrington's theorem). As discussed in Section 4.2, this extension is especially apt for algorithms such as block ciphers, which maintain and update a small "working state" and read off a (multi-bit) output from that state at the end.

1.2 Our Techniques

We now give an overview of our techniques, and explain how they relate to other known approaches. To keep the presentation simple, we describe our techniques in terms of *keyed* arithmetic circuit families $C : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$, as described in Section 2.4. (We note that we can obtain keyed circuit families from various other machine models, including general Boolean circuits, by the universal-program transformations of Section 2.4.)

[3] Indeed, the negative result of [BGI+01] for VBB in the standard model is based on an attack in which an obfuscated circuit is evaluated on its own bit representation, which of course depends fundamentally on the concrete instantiation of multilinear maps.

[4] For simplicity, we restrict our discussion here to *keyed* circuit families (Section 2.4), as discussed above.

Known approaches. In general, to construct program obfuscation, the essential task is the following. The evaluator, using the obfuscated circuit $\mathcal{O}(C(\cdot,\mathbf{y}))$, should be able to evaluate $C(\cdot,\mathbf{y})$ on inputs $\mathbf{x} \in \{0,1\}^n$ of his choice in some "hidden" representation, revealing only the result $C(\mathbf{x},\mathbf{y})$. Fully homomorphic encryption (FHE) gets us halfway there: while we can encrypt the secret key $\mathbf{y}$, so that the adversary can encrypt an input $\mathbf{x}$ and evaluate $C(\mathbf{x},\mathbf{y})$ homomorphically, the adversary can only learn an encryption of $C(\mathbf{x},\mathbf{y})$, not the output $C(\mathbf{x},\mathbf{y})$ itself. In all known constructions of general program obfuscation (including this work), this problem is solved via *multilinear maps* [BS03, GGH13a, CLT13, GGH14].

Multilinear maps, also known as graded encodings or graded multilinear maps [GGH13a, CLT13, GGH14], are a generalization of bilinear maps such as pairings over elliptic curves [Mil04, MOV93, Jou00, BF01]. Roughly speaking, a multilinear map lets us take a scalar x and produce an encoded version, $\hat{x} = [x]_S$, where $S \subseteq \mathcal{U}$ is a multi-set, called an *index set*, that indicates the level of the encoding $\hat{x}$ in a given hierarchy (namely, the subsets of $\mathcal{U}$ ordered by inclusion).[5] Elements can be added within the same index set, $[x]_S + [y]_S = [x+y]_S$; and elements can be multiplied, $[x]_S \cdot [y]_T = [xy]_{ST}$, as long as the resulting index set ST is still contained in $\mathcal{U}$. Finally, elements encoded at $\mathcal{U}$ itself can be zero-tested, to determine whether they encode the scalar 0. (For a more detailed description of multilinear maps we refer the reader to Section 2.5.)

Intuitively, multilinear maps seem like a perfect fit for program obfuscation. If we give out encoded versions of the secret key input $\mathbf{y} \in \{0,1\}^m$, then the evaluator can encode $\mathbf{x} \in \{0,1\}^m$ himself, use the multilinear map's arithmetic operations to evaluate C on the encoded elements, and zero-test the result to determine the output $C(\mathbf{x},\mathbf{y}) \in \{0,1\}$. Unfortunately, unless we are extremely careful, the adversary can also evaluate other circuits $C'(\mathbf{x},\mathbf{y}) \neq C(\mathbf{x},\mathbf{y})$ on the encoded inputs—such as the circuit C' that ignores the input $\mathbf{x}$ and leaks a bit of the secret key $\mathbf{y}$. Previously known approaches [GGH$^+$13b, BR14, BGK$^+$14, AGIS14, GLSW14] solve this problem by "garbling" the program $C(\cdot,\mathbf{y})$, converting it to a randomized matrix branching program via Kilian's protocol [Kil88].

Structure of our scheme. In our construction, we do not convert the circuit $C(\cdot,\mathbf{y})$ to a matrix branching program. Rather, evaluation of the obfuscated circuit $\mathcal{O}(C(\cdot,\mathbf{y}))$ follows the structure of the original circuit C, perfoming C's operations on encoded versions of $\mathbf{x},\mathbf{y}$ in the multilinear map (as depicted in Figure 1). To make sure the adversary evaluates the correct circuit, we make essential use of *composite-order* multilinear maps such as the CLT scheme [CLT13]. We encode scalars in $\mathbb{Z}_N$ for a composite modulus $N = N_{\text{ev}}N_{\text{chk}}$, and we view $\mathbb{Z}_N$ as a direct product of the two rings $\mathbb{Z}_{N_{\text{ev}}}, \mathbb{Z}_{N_{\text{chk}}}$, defined by the Chinese Remainder Theorem. To emphasize this intuition, we write $[x_1,x_2]_S$ to refer to an encoding, at index set S (Section 2.5), of the value $x \in \mathbb{Z}_N$ such that $x \equiv x_1 \pmod{N_{\text{ev}}}$ and $x \equiv x_2 \pmod{N_{\text{chk}}}$. Evidently the multilinear map operations $(+,\times)$ operate componentwise on these pairs, and a value encodes zero only if both components are zero.

Now, in our construction, the second component of the direct product ($\mathbb{Z}_{N_{\text{chk}}}$) serves as a kind of "checksum" for the adversary's evaluation. When the adversary aims to learn the value of some other circuit $C'(x_1,\ldots,x_n,y_1,\ldots,y_m)$, he will be forced to evaluate the same polynomial in parallel (in the second component), on the uniformly random values $\alpha_1,\ldots,\alpha_n,\beta_1,\ldots,\beta_m$, as depicted in Figure 1. At the end of this procedure, we also provide a "check" encoding $\hat{C}^*$, whose $\mathbb{Z}_{N_{\text{chk}}}$ component is the *precomputed* value $C(\alpha_1,\ldots,\alpha_n,\beta_1,\ldots,\beta_m)$. The structure of our scheme ensures (roughly speaking) that the adversary can only perform a zero-test by subtracting off a multiple of this encoding $\hat{C}^*$. (For more details, we refer the reader to Section 3.1.)

[5] We describe here the case of *asymmetric* multilinear maps, since this is the one relevant to our constructions in this work.

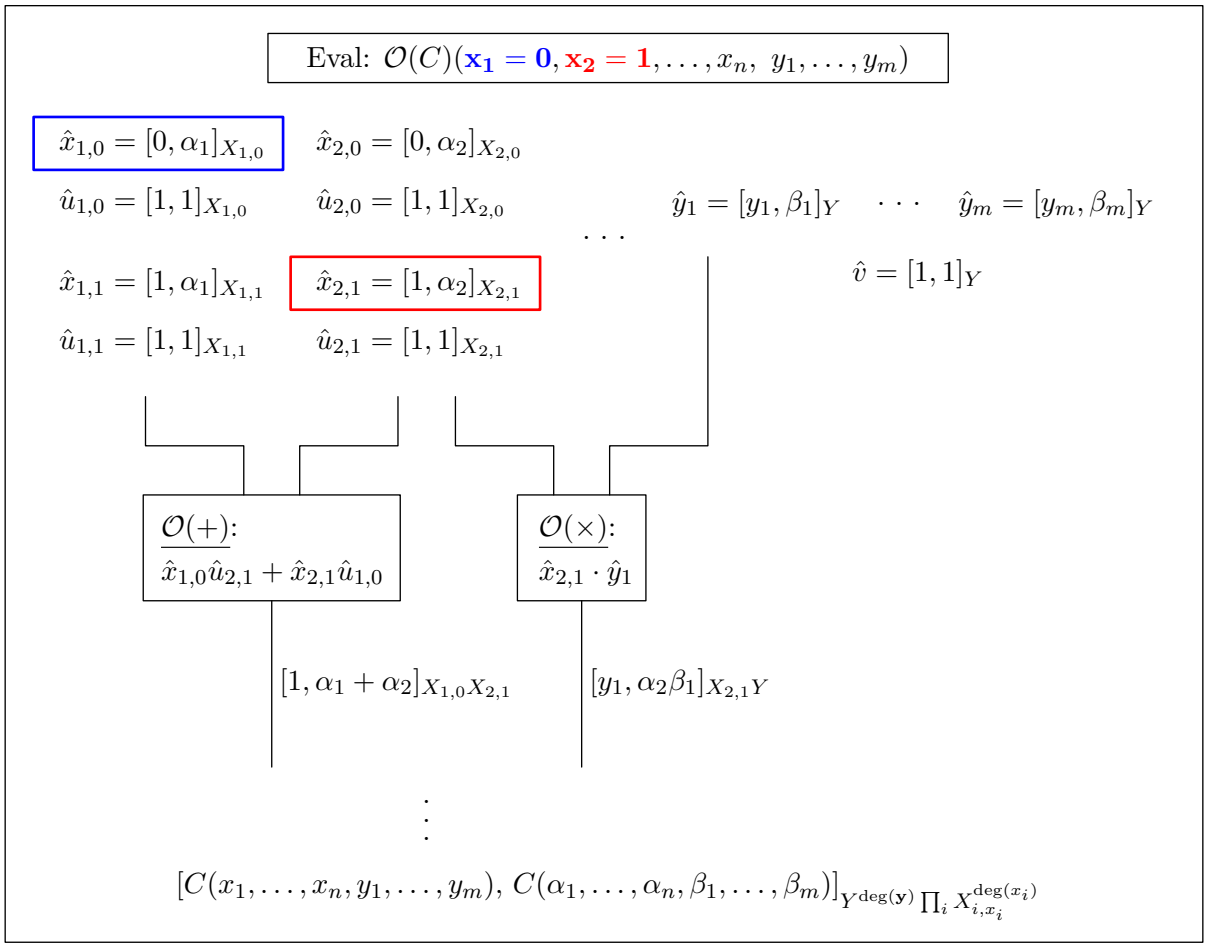

Figure 1: The first step of our evaluation procedure, for an obfuscated (keyed) arithmetic circuit. First, we use the bits of the input string $\mathbf{x}$ (e.g., $x_1 = 1$, $x_2 = 0$, ..., x_n) to select the relevant input encodings $\hat{x}_{1,1}, \hat{x}_{2,0}, \ldots, \hat{x}_n$. We then run C directly on the encodings $\hat{x}_{1,1}, \hat{x}_{2,0}, \ldots, \hat{y}_1, \ldots, \hat{y}_m$, implementing C's arithmetic operations via the multilinear map, and multiplying by encodings of 1 to make index sets match. (Here $\deg(x_i)$ is the degree of C, as a multivariate polynomial, in the variable x_i; and similarly $\deg(\mathbf{y})$ is the total degree of C in the variables $y_1, \ldots, y_m$.)

This design ensures that the adversary will learn nothing from evaluating the wrong circuit. Regardless of the inputs $\mathbf{x}, \mathbf{y}$, if the adversary evaluates an incorrect expression $C' \not\equiv C$, the result will not match our precomputed value $C(\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_m)$ modulo N_{chk}, and hence the final subtraction will produce a nonzero value modulo $N = N_{\text{ev}} N_{\text{chk}}$ (so that the multilinear map's zero-test operation always returns "nonzero"). In essence, we have forced the adversary to run the Schwartz-Zippel identity-testing algorithm on his own chosen expression C', in parallel (componentwise) with its actual evaluation on $x_1, \ldots, x_n, y_1, \ldots, y_m$.

Enforcing consistency: index sets with multiplicity. In addition to making sure the adversary cannot evaluate the wrong circuit $C' \neq C$, we must also defend against "mix-and-match" attacks, in which the adversary evaluates the correct circuit C, but uses inconsistent values of input bits at different points in the evaluation. Since we do not convert every circuit to a branching program, it is not clear how to solve this problem with the index set constraint techniques of [BR14, BGK+14]. In our model, the adversary must be allowed plenty of flexibility in construct-

ing his chosen query (since the honest evaluation follows the structure of the original circuit C, which is arbitrary), and yet the adversary must be able to complete all (and only) the consistent evaluations to the top-level index set $\mathcal{U}$.

Instead, we propose the following approach, depicted in Figure 1. We encode each input bit $(\hat{x}_{1,0}, \hat{x}_{1,1}, \hat{x}_{2,0}, \ldots)$ at its own singleton index set $(X_{1,0}, X_{1,1}, X_{2,0}, \ldots)$. The adversary can evaluate whatever expressions he chooses, and the associated index sets will track the degree of the expression in each variable. Then, we give out "interlocking" elements $\hat{z}_{i,b}$ whose index sets contain $X_{i,1-b}^{\deg(x_i)}$ for each bit choice $b \in \{0,1\}$ (where $\deg(x_i)$ is the degree of the variable x_i in the actual circuit C). By design of the index sets (Section 3), the adversary is forced to incorporate these elements $\hat{z}_{i,b}$ into any monomial that reaches the top level $\mathcal{U}$; but their index sets prevent the adversary from making any input-inconsistent choices within a given monomial. This, in turn, lets us decompose the adversary's queries into subqueries each consistent with one input $\mathbf{x} \in \{0,1\}^n$, which suffices for our purposes in the security proof.

Enforcing sequentiality: straddling sets and commitments. In order to achieve virtual black-box (VBB) security (in the generic model), our construction must also address the following subtle issue, raised in [BR14, BGK+14]. Roughly speaking, an efficient simulator in the generic model must examine the arithmetic expression z that the adversary evaluates via the multilinear map operations, and determine whether z would evaluate to zero in the real scheme. The simulator must make this decision based only on the information it receives from its own oracle $C(\cdot, \mathbf{y})$, which means that if the expression z includes terms from superpolynomially many possible inputs $\mathbf{x}$, then the simulator cannot necessarily answer the query efficiently.

We solve this problem by adapting an elegant technique of Barak et al. [BGK+14]. In that work, the authors describe a tool called *straddling sets*. A straddling set system consists of two partitions $\mathcal{S}_0, \mathcal{S}_1$ of the set $[n]$, each consisting of $O(n)$ subsets. The subsets are arranged so that once we choose a set from (say) the partition $\mathcal{S}_0$, we have committed to $\mathcal{S}_0$, and we cannot complete this set to form a full partition of $[n]$ except by adding all (and only) the remaining sets in the partition $\mathcal{S}_0$. The construction of [BGK+14] associates a straddling set system to each input bit $i \in \{1, \ldots, n\}$, for a total of $O(n^2)$ sets among all n partitions, and the index set of each encoded matrix includes a set from each of two different straddling set systems, indicating which of the corresponding two input bits the matrix selects (in the matrix branching program). Our use of straddling sets in this work is similar to their use in [BGK+14], with some adaptations to restrict which of our terms induce which straddling-set dependencies. We defer the full details to Construction 3.1.

1.3 Related Work

As discussed above, our work builds on earlier constructions of program obfuscation [GGH+13b, CV13, BR14, BGK+14, AGIS14, GLSW14], but our new techniques differ in multiple ways—most notably, we obfuscate circuits directly, without converting them to branching programs.

The work of Gentry et al. [GLSW14] constructs indistinguishability obfuscation ($i\mathcal{O}$) from composite-order multilinear maps. In that work, extending the techniques of [GLW14], the authors show that $i\mathcal{O}$ can be based on a single, falsifiable assumption, independent of the particular circuit to be obfuscated. Previously it was only known how to prove $i\mathcal{O}$ in generic models of multilinear maps [GGH+13b, BR14, BGK+14], or from meta-assumptions that quantify over many circuits [PST14]. In [GLSW14], the emphasis is on the new assumption; the main construction is based on the standard paradigm of converting circuits to branching programs, as in [GGH+13b, BR14, BGK+14]. By contrast, our work proposes a new kind of construction, which avoids branching programs entirely; while our security proof is given in a generic model similar

to that of [BGK+14]. Thus, our work is largely orthogonal to that of [GLSW14]. As discussed above, we believe it may be possible to adapt our construction to base security on a single falsifiable assumption, as in [GLSW14], and we leave this as an important open problem for future work.

Our work is also complementary to that of Ananth et al. [AGIS14]. In that work, the authors give an obfuscation construction that is still based on matrix branching programs, as in [GGH+13b, BR14, BGK+14], but constructs those branching programs much more efficiently when the programs to be obfuscated are given as Boolean formulas. A key observation in [AGIS14] is that in order to evaluate a Boolean formula ϕ efficiently, we can simply test whether two specific vertices are connected in a directed graph related to ϕ. As the authors observe, this graph connectivity computation can be written as matrix multiplication, and thus it is well-suited to known approaches via matrix branching programs. More broadly, however, the graph connectivity computation is well-suited to program obfuscation *in general*—because the structure of matrix multiplication is independent of the input data (Section 2.2), and because it has relatively low degree as an arithmetic circuit. Indeed, the new obfuscator we develop in this work could also be run on the connectivity algorithms of [AGIS14]; and, as we will see in Section 4.3.3, for some parameter settings this would yield even better performance than running our obfuscator on a program that evaluates the formula ϕ directly (i.e., without converting it to a graph connectivity problem). The techniques we develop in this work expose a rich space of design choices for the computations that are *input* to the obfuscator, and the connectivity computation of [AGIS14] is an interesting example of one such design.

2 Preliminaries

2.1 Conventions

For integers n, a, b, we denote by $[n]$ the set $\{1, \ldots, n\}$, and by $[a, b]$ the set $\{a, \ldots, b\}$. For a finite set S, we write $\mathsf{Uniform}(S)$ to mean the probability distribution that is uniform over the elements of S. For integers a, b, we write $\mathsf{Primes}[a, b]$ to mean the set of all prime numbers in $[a, b]$, and we overload this notation to refer to the distribution $\mathsf{Uniform}(\mathsf{Primes}[a, b])$. We also assume various conventions of cryptography. Specifically, we define a variable λ, called the security parameter. We define a *negligible function* to be a function $\varepsilon(\lambda)$ that is $o(1/\lambda^c)$ for every $c > 0$, and we write $\mathrm{negl}(\lambda)$ to denote a negligible function of n. We define an *efficient algorithm* to be a probabilistic polynomial-time Turing machine. We say an event occurs with *negligible probability* if the probability of the event is $\mathrm{negl}(\lambda)$, and an event occurs with *overwhelming probability* if its complement occurs with negligible probability.

2.2 Oblivious Computation and the "Mux" Operation

When we perform any computation on hidden data—not just in obfuscation, where the program is hidden, but also in settings such as homomorphic encryption and secure multi-party computation—we can distinguish between two main problems. First, we must ensure that each primitive operation (such as an addition, multiplication, or Boolean gate) does not leak information about hidden values, when implemented using the cryptographic scheme of choice. Second, we must ensure that the *identities* of the primitive operations performed do not leak information. While the solution to the first problem varies widely, we generally solve the second problem by transforming the computation so that the sequence of operations performed is independent of the input data. We call programs that satisfy this criterion *data-oblivious*, or *oblivious*.

Formally, this criterion has many variants; for a full exposition, we refer the reader to the work of Mitchell and Zimmerman [MZ14]. In this work, we will consider a program *oblivious* if the sequence of primitive operations performed, as well as the identities of their operands (e.g., registers or memory locations in a RAM) is a deterministic function solely of the input length, and does not depend on the input. In the language of [MZ14], this corresponds to the strongest definition, that of *data-independence*.

To make a program oblivious, there are many standard techniques. We now describe one such technique, known as "arithmetization" or "multiplexing" (abbreviated "mux"), which is involved in various compiler optimizations and static analyses of programs. The idea is very simple: whenever a program would call for input-dependent control flow, such as "if x then $y \leftarrow z$; else $y \leftarrow w$;", we remove the conditional, and replace every assignment statement in both branches with an arithmetized version: "$y \leftarrow x \cdot z + (1 - x) \cdot w$", also denoted "$y \leftarrow \mathrm{mux}(x, z, w)$". This ternary "mux" operation is crucial to removing input-dependent control flow from programs, and we use it extensively in our applications (Section 4).

2.3 Straight-Line Programs (Arithmetic Circuits)

In our obfuscation construction, we will find it natural to work with the computational model of straight-line programs over the integers. Formally, we will model these programs as follows.

Definition 2.1 (Straight-Line Program)**.** A *straight-line program* $P : \mathbb{Z}^n \to \mathbb{Z}$ is a sequence of statements:

$$\text{input } x_1, \ldots, x_n\,; \quad v_1 \leftarrow w_{1,1} \text{ op } w_{1,2}\,; \quad \ldots\,; \quad v_\ell \leftarrow w_{\ell,1} \text{ op } w_{\ell,2}\,; \quad \text{output } v_\ell$$

where $v_1, \ldots, v_\ell$ are variables (e.g., distinct bit strings), each instance of op is an arithmetic operator ($+$ or $\times$), and each $w_{1,1}, w_{1,2}, \ldots, w_{\ell,1}, w_{\ell,2}$ is either a variable in $v_1, \ldots, v_\ell$; an input in $x_1, \ldots, x_n$; or else a constant in $\{-1, 1\}$. For an input tuple $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{Z}^n$, we write $P(x_1, \ldots, x_n)$ to denote the result of substituting the values $x_1, \ldots, x_n$ for the corresponding inputs in P, performing the arithmetic operation corresponding to each instruction in sequence, and yielding the output v_ℓ.

We say a straight-line program $P : \mathbb{Z}^n \to \mathbb{Z}$ computes a Boolean function $f : \{0,1\}^n \to \{0,1\}$ if for all $\mathbf{x} \in \{0,1\}^n$, we have $f(\mathbf{x}) = 1 \Leftrightarrow P(\mathbf{x}) \neq 0$. When the context is clear, we abuse notation to write $P(\mathbf{x}) : \{0,1\}^n \to \{0,1\}$ to denote the Boolean function f that P computes.

The model of straight-line programs is extremely general. The execution of any machine—say, a Turing machine or RAM—can be expressed as a straight-line program over $\mathbb{Z}$, with overhead at most polylogarithmic,[6] provided that the machine is already oblivious (Section 2.2).

Remark 2.2 (Straight-Line Programs are Arithmetic Circuits)**.** Straight-line programs are naturally identified with arithmetic circuits. Formally, we consider arithmetic circuits with addition and multiplication gates over $\mathbb{Z}$, with fanin 2 and unbounded fanout. We also assume the circuits have implicit inputs representing the constants $\pm 1 \in \mathbb{Z}$, and thus they can form arbitrary integer constants (up to the limitations of circuit size and depth).

In this work, we will refer to straight-line programs and arithmetic circuits interchangeably.

An arithmetic circuit $C : \{0,1\}^n \to \{0,1\}$ can also be expressed as a multivariate polynomial in $\mathbb{Z}[x_1, \ldots, x_n]$ (perhaps after duplicating gates to account for fanout), and we identify circuits with their corresponding polynomials. Although the polynomial for a given circuit C may be of exponential size, it can still be evaluated efficiently, and we can perform algebraic substitutions

[6] For instance, in some models there is overhead involved in decomposing word operations into bits.

on it. We define the *degree* of an arithmetic circuit C in each input variable as the degree of its corresponding polynomial in that variable, and similarly for the total degree. By convention, we always refer to the *formal* degree, counting monomials whose coefficients are zero. We note that a circuit of depth d can have degree at most 2^d, since each of its gates has fanin two.

Given a Boolean circuit C, evidently we can convert it into an arithmetic circuit C' that computes the same function, with at most a constant factor overhead both in size and in depth. For instance, we can express C in terms of NAND gates, and replace each NAND gate in C by the polynomial $(r, s) \mapsto 1 - rs$ in the arithmetic circuit C'. In Section 4, we will also consider more efficient transformations. From here on, we will assume that programs to be obfuscated can be written as straight-line programs (i.e., arithmetic circuits), as we have just described.

2.4 Keyed Programs

In many cryptographic applications of obfuscation, we do not depend on hiding the entire structure of the obfuscated program from the adversary, but rather only need to hide a short secret key embedded in the program. We can formalize this notion as follows.

Definition 2.3 (Keyed Circuit Family)**.** Let $C : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$ be an arithmetic circuit of size s and depth d, and for each $\mathbf{y} \in \{0,1\}^m$, define the function $f_{\mathbf{y}}(\mathbf{x}) = C(\mathbf{x}, \mathbf{y})$ for all inputs $\mathbf{x} \in \{0,1\}^n$. If $(C_{\mathbf{y}})_{\mathbf{y}\in\{0,1\}^m}$ is a family of arithmetic circuits such that each $C_{\mathbf{y}}$ computes $f_{\mathbf{y}}$, then we say that $(C_{\mathbf{y}})_{\mathbf{y}\in\{0,1\}^m}$ is a *keyed circuit family*, of size s and depth d, corresponding to the universal circuit C.

The model of "keyed" programs is especially natural for obfuscation, and we expect that in most cryptographic applications, it will find more use than general-purpose obfuscation. For theoretical purposes, however, we would still like to construct general-purpose obfuscation for large classes of circuits such as NC^1 or P/poly, for which the obfuscation must hide everything except the size of the circuit to be obfuscated.

Thus, we now provide standard transformations from general (unkeyed) circuit families to keyed circuit families. Intuitively, in these transformations the secret key $\mathbf{y} \in \{0,1\}^m$ represents the entire circuit to be obfuscated, and the keyed circuit C is a universal circuit.

Construction 2.4 (Universal Straight-Line Program)**.** Regard the circuit as an (arithmetic) straight-line program (Section 2.3) consisting of ℓ instructions. For each $k \in [\ell]$, replace the k^{th} instruction "$v_k \leftarrow w_{k,1}$ op $w_{k,2}$" as follows. First, define new secret key input bits $(y_{k,1,i}, y_{k,2,i})_{i\in[k-1]}$. Generate new variables $w'_{k,1} \leftarrow 0$, $w'_{k,2} \leftarrow 0$, and generate $2(k-1)$ "mux" operations (Section 2.2) with the outputs of all previous instructions, as follows:

$$w'_{k,1} \leftarrow \mathrm{mux}(y_{k,1,1}, w'_{k,1}, v_i); \quad \ldots; \quad w'_{k,1} \leftarrow \mathrm{mux}(y_{k,1,k-1}, w'_{k,1}, v_{k-1})$$

$$w'_{k,2} \leftarrow \mathrm{mux}(y_{k,2,1}, w'_{k,2}, v_i); \quad \ldots; \quad w'_{k,2} \leftarrow \mathrm{mux}(y_{k,2,k-1}, w'_{k,2}, v_{k-1})$$

Generate the following statements for $\mathrm{op} \in \{+, -, *\}$:

$$w_{k,\mathrm{op}} \leftarrow w'_{k,1} \text{ op } w'_{k,2}$$

Finally, define new secret key input bits $y_{k,-}, y_{k,*}$, and use the "mux" operation again to merge the results as follows:

$$w_k \leftarrow w_{k,+}; \quad w_k \leftarrow \mathrm{mux}(y_{k,-}, w_k, w_{k,-}); \quad w_k \leftarrow \mathrm{mux}(y_{k,*}, w_k, w_{k,*})$$

Since Construction 2.4 emits $(k-1)$ instructions in place of the k^{th} instruction of the original program, the size of the resulting program is quadratic in that of the original (disregarding polylogarithmic factors from writing down names of variables).

Construction 2.4 works best for circuits whose depth is at least polylogarithmic in their size, since then the additional depth from the mux operations is relatively insignificant. For completeness, we also provide a transformation tailored to low-depth (low-degree) circuits. Notably, this second transformation results in a circuit whose size is exponential in the depth of the original circuit (like the transformation via Barrington's Theorem), and hence applies only to NC^1. However, for these circuits, unlike Construction 2.4, Construction 2.5 preserves the depth up to constant factors.

Construction 2.5 (Universal Boolean Formula)**.** We give only a sketch here; we defer the details to Section 4.3.2. Using DeMorgan's laws, translate the circuit to an equivalent monotone circuit in its inputs and their negations (i.e., only using binary AND and OR gates). Duplicate gates as necessary to achieve fanout 1, and add dummy gates to make the circuit *balanced*, i.e., a full binary tree of depth $O(d)$. In place of each of the circuit's input wires, add a tree of "mux" operations (Section 2.2) to select one of the n original inputs, as determined by secret key input bits. Finally, convert each Boolean gate to an arithmetic function, hiding the gate's identity by deriving some of the function's inputs from secret key input bits.

We emphasize again that the transformations of this section are mainly for theoretical purposes. In practice, a much better approach would be to design, for each desired cryptographic application of obfuscation, a restricted family of circuits (much smaller than P/poly) that is already keyed with respect to the particular secret that needs to be hidden.

2.5 Composite-Order Multilinear Maps

Multilinear maps [BS03], also known as graded multilinear maps or graded encodings [GGH13a, CLT13, GGH14], are a generalization of bilinear maps such as pairings over elliptic curves [Mil04, MOV93, Jou00, BF01]. Intuitively, a multilinear map lets us take scalars x, y and produce corresponding encodings $\hat{x}, \hat{y}$ at any level of a given hierarchy, so that we can still perform arithmetic operations (e.g., $x+y, xy$) on the encoded representations, and yet it is hard to recover the original scalars x, y from encodings $\hat{x}, \hat{y}$. [7] For example, in a symmetric bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ (where g generates $\mathbb{G}$, and $e(g,g)$ generates $\mathbb{G}_T$), a scalar $x \in \mathbb{Z}$ can be encoded in $\mathbb{G}$ as g^x, or encoded in $\mathbb{G}_T$ as $e(g,g)^x$. The levels of the hierarchy here are $\mathbb{G}$ and $\mathbb{G}_T$, and the hierarchy's structure enforces constraints on the arithmetic operations that we can perform. For instance, via the group operation we can compute g^{x+y} (an encoding of $x+y$) from g^x and g^y (encodings of x and y), but to obtain an encoding of xy, we must increase the level in the hierarchy from $\mathbb{G}$ to $\mathbb{G}_T$, by computing the pairing $e(g^x, g^y) = e(g,g)^{xy}$.

In the case of symmetric bilinear maps, this hierarchical structure can be identified with the integers $0, 1, 2$ as indices, where the index 0 represents scalars, 1 represents elements of $\mathbb{G}$, and 2 represents elements of $\mathbb{G}_T$. Elements at the same index can be added together, while elements at arbitrary indices can be multiplied, but their indices add. For asymmetric bilinear maps, the more natural analogy is that of a subset lattice: specifically, a map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is identified with

[7] In fact, as we will see below, we usually assume more than this: recovering x from the encoding $\hat{x}$ corresponds to solving the discrete-log problem, while intuitively we expect that the adversary cannot do anything useful with $\hat{x}$, except apply the permitted arithmetic operations and test the result for equality against other encodings. In Section 2.6, we will make this intuition more precise.

the subset lattice $\emptyset \subseteq \{A\},\{B\} \subseteq \{A, B\}$, where $\emptyset$ corresponds to scalars, $\{A\}$ to $\mathbb{G}_1$, $\{B\}$ to $\mathbb{G}_2$, and $\{A, B\}$ to $\mathbb{G}_T$.

More generally, in the case of asymmetric multilinear maps (which permit more than two sequential multiplications of encoded elements), it is standard to work with general subset lattices, where the sets may contain elements with multiplicity. By convention, we will say that these sets are made up of *formal symbols*, denoted by capital letters (A, B, C), which serve the same role as formal variables in polynomials. Formally, we state the following definitions.

Definition 2.6 (Formal Symbol)**.** A *formal symbol* is a bit string in $\{0,1\}^*$, and distinct variables denote distinct bit strings. A *fresh* formal symbol is any bit string in $\{0,1\}^*$ that has not already been assigned to another formal symbol.

Definition 2.7 (Index Sets)**.** An *index set* is a multi-set of formal symbols called *indices*. The multiplicity of each index is written in binary, and so the degree of an set may be up to exponential in the size of its representation. By convention, for index sets we use set notation and product notation interchangeably, so that A^3BC^2 represents $\{A, A, A, B, C, C\}$, and $A^3BC^2 \cup ABC = A^4B^2C^3$.

Definition 2.8 (Composite-Order Multilinear Map ([BS03, GGH13a, CLT13, GLW14], adapted))**.** A *composite-order multilinear map* supports the following operations. Each operation ($\mathsf{CM.Setup}$, $\mathsf{CM.Add}$, $\mathsf{CM.Mult}$, $\mathsf{CM.ZeroTest}$, $\mathsf{CM.Encode}$) is implemented by an efficient randomized algorithm.

- The setup procedure receives as input an index set $\mathcal{U}$ (Definition 2.7), which we refer to as the "top-level index set", as well as the security parameter λ (in unary), and an integer k indicating the number of factors to generate for the modulus. It produces public parameters pp, secret parameters sp, and integers $N_1, \ldots, N_k$ as follows:

$$\mathsf{CM.Setup}(\mathcal{U}, 1^\lambda, k) \;\rightarrow\; (\mathsf{pp},\ \mathsf{sp},\ N_1, \ldots, N_k)$$

  Each integer $N_1, \ldots, N_k$ is a product of $\mathrm{poly}(\lambda)$ primes, and each of these $k \cdot \mathrm{poly}(\lambda)$ primes is drawn independently from $\mathsf{Primes}[2^\lambda, 2^{\lambda+1}]$. We also define $N = \prod_{i \in [k]} N_i$, the overall modulus.[8]

- For each index set $\mathcal{S} \subseteq \mathcal{U}$, and each scalar $x \in \mathbb{Z}_N$, there is a set of strings $[x]_\mathcal{S} \subseteq \{0,1\}^*$, i.e., the set of all valid encodings of x at index set $\mathcal{S}$. [9] From here on, we will abuse notation to write $[x]_\mathcal{S}$ to stand for any element of $[x]_\mathcal{S}$ (i.e., any valid encoding of x at the index set $\mathcal{S}$).

- Elements at the same index set $\mathcal{S} \subseteq \mathcal{U}$ can be added, with the result also encoded at $\mathcal{S}$:

$$\mathsf{CM.Add}(\mathsf{pp},\ [x]_\mathcal{S},\ [y]_\mathcal{S}) \;\rightarrow\; [x+y]_\mathcal{S}$$

- Elements at two index sets $\mathcal{S}_1, \mathcal{S}_2$ can be multiplied, with the result encoded at the union of the two sets, as long as their union is still contained in $\mathcal{U}$:

$$\mathsf{CM.Mult}(\mathsf{pp},\ [x]_{\mathcal{S}_1},\ [y]_{\mathcal{S}_2}) \;\rightarrow\; \begin{cases} [xy]_{\mathcal{S}_1 \cup \mathcal{S}_2} & \text{if } \mathcal{S}_1 \cup \mathcal{S}_2 \subseteq \mathcal{U} \\ \perp & \text{otherwise} \end{cases}$$

[8] We remark here that our construction does not rely on the individual moduli $N_1, \ldots, N_k$ being composite, but we present the model in this full generality since it may be required in the chosen concrete instantiation, such as in the CLT multilinear map [CLT13].

[9] To be precise, we define $[x]_\mathcal{S} = \{\chi \in \{0,1\}^* : \mathsf{CM.IsEncoding}(\mathsf{pp}, \chi, x, \mathcal{S})\}$, where the predicate $\mathsf{CM.IsEncoding}$ is specified by the concrete instantiation of the multilinear map. The predicate $\mathsf{CM.IsEncoding}$ need not be efficiently decidable—and indeed, for the security of the multilinear map, it should not be.

- Elements at the top level $\mathcal{U}$ can be *zero-tested*:

$$\mathsf{CM.ZeroTest}(\mathsf{pp},\ [x]_{\mathcal{S}}) \ \rightarrow \ \begin{cases} \text{“zero”} & \text{if } \mathcal{S} = \mathcal{U} \text{ and } x = 0 \in \mathbb{Z}_N \\ \text{“nonzero”} & \text{otherwise} \end{cases}$$

- Using the secret parameters, one can generate a representation of a given scalar $x \in \mathbb{Z}$ at any index set $\mathcal{S} \subseteq \mathcal{U}$:

$$\mathsf{CM.Encode}(\mathsf{sp},\ x,\ \mathcal{S}) \ \rightarrow \ [x]_{\mathcal{S}}$$

- For the trivial index set $\mathcal{S} = \emptyset$, we specify that the valid encodings $[x]_\emptyset$ are just the integers congruent to x modulo N. (So, for instance, we can perform subtraction via $\mathsf{CM.Add}$, by scalar multiplication with -1.)

By convention (and by analogy to the setting of symmetric multilinear maps), we refer to the total degree of $\mathcal{U}$ as the *degree of multilinearity* of the map. When the context is clear, we also abuse notation to write, for encodings $\hat{a}, \hat{b}$, the expression $\hat{a} + \hat{b}$ to mean $\mathsf{CM.Add}(\mathsf{CM.pp}, \hat{a}, \hat{b})$; the expression $\hat{a}\hat{b}$ to mean $\mathsf{CM.Mult}(\mathsf{CM.pp}, \hat{a}, \hat{b})$; and likewise for other arithmetic expressions.

Features of composite order. By analogy to composite-order bilinear groups [BGN05], we would expect that composite-order multilinear maps would be significantly more powerful than their traditional prime-order analogs. Intuitively, this power is due to the fact that by encoding integers in $\mathbb{Z}_N$ for composite $N = N_1 \cdots N_k$, we implicitly encode a direct product, $\mathbb{Z}_{N_1} \times \ldots \times \mathbb{Z}_{N_k}$, as defined by the Chinese Remainder Theorem. Each of the k components can be used to store useful information, on which the ring operations act componentwise, and a value will pass the multilinear map's zero-test only if it encodes zero in *every* component (i.e., modulo every N_i). Without knowing the factorization, however, the adversary cannot easily eliminate one component of an encoded value without eliminating them all. To better express this intuitive view, we introduce the following notation.

Remark 2.9 (Notation for Encodings of Direct Products)**.** We write $[x_1, x_2, \ldots, x_k]_S$ to refer to an encoding, at index set S, of the value $x \in \mathbb{Z}_N$ such that $x \equiv x_i \pmod{N_i}$ for each $i \in [k]$ (as determined by the Chinese Remainder Theorem).

2.6 The Generic Multilinear Map Model

To define security for composite-order multilinear maps, we will operate in a *generic model* of composite-order multilinear maps, which generalizes existing generic models for the prime-order case [GGH$^+$13b, BR14, BGK$^+$14]. This model is similar to the generic group model [Sho97]: intuitively, in the generic model, the only thing an adversary can do with encoded ring elements is to apply the operations of the multilinear map.

More precisely, we say a scheme that uses multilinear maps is “secure in the generic model” if, for any concrete adversary breaking the real scheme, there is a generic adversary breaking a modified scheme in which the encoded ring elements are replaced by “handles” (concretely, fresh nonces), which the generic-model adversary can supply to a stateful oracle $\mathcal{M}$ (which performs the corresponding ring operations internally). We define the oracle $\mathcal{M}$ formally as follows.

Definition 2.10 (Generic Multilinear Map Oracle ([GGH$^+$13b, BR14, BGK$^+$14], adapted))**.** A *generic multilinear map oracle* is a stateful oracle $\mathcal{M}$ that responds to queries as follows.

- On a query $\mathsf{CM.Setup}(\mathcal{U}, 1^\lambda, k)$, the oracle will generate integers $N_1, \ldots, N_k$ as in the real setup procedure (Definition 2.8), generate pp, sp as fresh nonces (i.e., distinct from any previous choices) uniformly at random from $\{0,1\}^\lambda$, and return $(\mathsf{pp}, \mathsf{sp}, N_1, \ldots, N_k)$. It will also store the inputs and the values generated, initialize an internal table $T \leftarrow \{\}$ (to store "handles", as described below), and set internal state so that subsequent $\mathsf{CM.Setup}$ queries fail.

- On a query $\mathsf{CM.Encode}(k,\ x,\ \mathcal{S})$, where $k \in \{0,1\}^\lambda$ and $x \in \mathbb{Z}$, the oracle will check that $k = \mathsf{sp}$ and $\mathcal{S} \subseteq \mathcal{U}$ (returning $\perp$ if the check fails). If the check passes, the oracle will generate a fresh nonce ("handle") $h \leftarrow \mathsf{Uniform}(\{0,1\}^\lambda)$, add the entry $h \mapsto (x, \mathcal{S})$ to the table T, and return h.

- On a query $\mathsf{CM.Add}(k, h_1, h_2)$, where $k, h_1, h_2 \in \{0,1\}^\lambda$, the oracle will check that $k = \mathsf{pp}$, and that the handles h_1, h_2 are present in its internal table T, and are mapped to values, resp., $(x_1, \mathcal{S}_1)$ and $(x_2, \mathcal{S}_2)$ such that $\mathcal{S}_1 = \mathcal{S}_2 = \mathcal{S} \subseteq \mathcal{U}$ (returning $\perp$ if the check fails). If the check passes, the oracle will generate a fresh handle $h \leftarrow \mathsf{Uniform}(\{0,1\}^\lambda)$, add the entry $h \mapsto (x_1 + x_2, \mathcal{S})$ to the table T, and return h.

- On a query $\mathsf{CM.Mult}(k, h_1, h_2)$, where $k, h_1, h_2 \in \{0,1\}^\lambda$, the oracle will check that $k = \mathsf{pp}$, and that the handles h_1, h_2 are present in its internal table T, and are mapped to values, resp., $(x_1, \mathcal{S}_1)$ and $(x_2, \mathcal{S}_2)$ such that $\mathcal{S}_1 \cup \mathcal{S}_2 \subseteq \mathcal{U}$ (returning $\perp$ if the check fails). If the check passes, the oracle will generate a fresh handle $h \leftarrow \mathsf{Uniform}(\{0,1\}^\lambda)$, add the entry $h \mapsto (x_1 x_2, \mathcal{S}_1 \cup \mathcal{S}_2)$ to the table T, and return h.

- On a query $\mathsf{CM.ZeroTest}(k, h)$, where $k, h \in \{0,1\}^\lambda$, the oracle will check that $k = \mathsf{pp}$, and that the table T contains an entry $h \mapsto (x, \mathcal{U})$ (immediately returning $\perp$ if the check fails). If the check passes, the oracle will return "zero" if $x \equiv 0 \pmod{N = N_1 \cdots N_k}$, and "nonzero" otherwise.

Remark 2.11 (Oracle Queries Referring to Formal Polynomials)**.** Although the generic multilinear map oracle is defined formally in terms of "handles" (Definition 2.10), it is usually more intuitive to regard each oracle query as *referring* to a formal query polynomial. The formal variables are specified by the expressions initially supplied to the $\mathsf{CM.Encode}$ procedure (as determined by the details of the construction), and the adversary can construct terms that refer to new polynomials by making oracle queries for the generic-model ring operations $\mathsf{CM.Add}$, $\mathsf{CM.Mult}$. Rather than operating on a "handle", then, each valid $\mathsf{CM.ZeroTest}$ query refers to a formal query polynomial[10] encoded at the top-level index set $\mathcal{U}$. The result of the query is "zero" precisely if the polynomial evaluates to zero, when its variables are instantiated with the joint distribution over their values in $\mathbb{Z}_N$, generated as in the real security game. The formal definitions required to justify this language are quite tedious, and we defer them to Appendix B.

To illustrate the language of Remark 2.11, we take the following example. Suppose that in some security game, the adversary receives values that were generated as $\hat{x} \leftarrow \mathsf{CM.Encode}(\mathsf{sp}, 2, S)$ and $\hat{y} \leftarrow \mathsf{CM.Encode}(\mathsf{sp}, 3, T)$. Then the formal variables of the construction are $\hat{x}$ and $\hat{y}$; and the adversary could submit a zero-test query that refers to the formal polynomial $\hat{x}\hat{y}$ (at the index set $\mathcal{U} = ST$). The real value of $\hat{x}\hat{y}$ is $2 \cdot 3 = 6 \neq 0 \in \mathbb{Z}$, since this is the value that would be in the oracle's table in the real game (if the adversary had made the corresponding query by invoking $\mathsf{CM.Mult}$ on the handles that refer to $\hat{x}, \hat{y}$). Thus, the result of the zero-test on the formal query polynomial $\hat{x}\hat{y}$ would be "nonzero".

[10] To represent a query polynomial concretely, we can use an arithmetic circuit—and thus, for instance, we can still perform efficient manipulations on query polynomials that have been subjected to repeated squaring.

Remark 2.12 (Unique Encodings and Zero-Testing)**.** In this work, as in [BGK+14], our generic model allows the adversary to zero-test *only* at the top-level index set $\mathcal{U}$. In all known multilinear map constructions, including our suggested instantiation via the CLT scheme [CLT13], no weaknesses are known that would permit zero-testing outside $\mathcal{U}$. However, if in the future we discover multilinear maps for which this operation is possible—for instance, if elements have unique encodings—then our obfuscation construction would need to be modified for this setting; we leave the details of this extension for future work. We also remark that even "clean" multilinear maps (Section 2.7) need not have unique encodings or support zero-testing outside $\mathcal{U}$.

Composite order in the generic model. We note that for composite-order multilinear maps in particular, formulation of the model requires some care. In generic models for groups of prime order p, the adversary is usually given p, which is reflected in the fact that the adversary's query polynomials (Remark 2.11) have coefficients over $\mathbb{Z}_p$ (see, e.g., the prime-order models of [GGH+13b, BR14, BGK+14]). In the composite-order setting, however, it is vital that the adversary cannot act independently on different components of our direct product encodings in $\mathbb{Z}_N$ (Remark 2.9). If the adversary knows the factorization of N, then he can do this easily, since then he can multiply an encoding by scalars corresponding to one or more of $N_1, \ldots, N_k$.

We might hope that in the generic model, we could still give the adversary the overall modulus N, relying on the hardness of factoring to rule out these projection queries. Unfortunately, given N, there may be many other ways for the adversary to construct query polynomials with undesired effects on our encodings in $\mathbb{Z}_N$. The only way we know to rule out this possibility is to make N itself unknown. In other words, we work in a ring of hidden order, which is formally reflected in the fact that the adversary's query polynomials (Remark 2.11) have coefficients over $\mathbb{Z}$ (rather than $\mathbb{Z}_N$). In this respect, our generic model for composite-order multilinear maps is similar to generic models in other composite-order settings [AM09].

2.7 "Noisy" and "Clean" Multilinear Maps

The abstract definition of multilinear maps (Definition 2.8) is very natural, but we still do not know whether it can be instantiated. The breakthrough work of Garg et al. [GGH13a] showed the first candidate construction of an *approximate* or "noisy" variant of multilinear maps, in which the representation of each encoded ring element includes a random error term. When ring elements are added or multiplied, the resulting error term increases; eventually, the noise overwhelms the signal, and the zero-testing procedure no longer recovers the correct answer. Thus, unlike the "clean" multilinear maps of Definition 2.8, known "noisy" multilinear maps include an *a priori* restriction of the number and types of operations that can be performed.

In known multilinear map constructions [GGH13a, CLT13, GGH14], the noise restriction behaves as follows. Each encoded ring element carries a noise bound. The result of a fresh encoding operation (via $\mathsf{CM.Encode}$) has a noise bound of $2^{f(\lambda)}$ (for some polynomial f pre-specified at setup); $\mathsf{CM.Add}$ results in a noise bound that grows with the sum of the bounds of its operands; and $\mathsf{CM.Mult}$ results in a noise bound that grows with the product. When the noise bound reaches $2^{g(\lambda)}$ (again for a pre-specified polynomial g), the zero-test operation always outputs $\perp$.

For our purposes in this work, we will model the noise restriction as stating that the multilinear map can only compute arithmetic expressions of *polynomial degree* (for a polynomial fixed at setup time)—or, equivalently, that the multiplicities of indices in the top-level index set $\mathcal{U}$ are presented in unary.

Definition 2.13 ("Noisy" Composite-Order Multilinear Map)**.** A *noisy composite-order multilinear map* is defined as in Definition 2.8, except that the top-level index set $\mathcal{U}$ has its multiplicities presented in unary.

We note that Definition 2.13 considers only the noise growth due to multiplication operations, and disregards that of addition operations.[11] Technically, in order to instantiate this definition with the CLT multilinear map [CLT13], we would also need to specify that the ring operations may fail for computations with many additions and very few multiplications. However, our main theorems are unaffected by this restriction. In a broader sense, we also find that this simple definition in terms of multilinearity degree is more natural, and is better suited to other potential approaches for instantiating multilinear maps (Section 2.8).

2.8 Instantiation of Multilinear Maps

As discussed above, we already know a candidate instantiation of "noisy" composite-order multilinear maps, via the CLT scheme [CLT13]. We now briefly recount the structure of this scheme. Fix a top-level index set $\mathcal{U} = A_1^{u_1} \cdots A_\ell^{u_\ell}$, where $A_1, \ldots, A_\ell$ are formal symbols (Definition 2.7). The CLT scheme generates an "inner" modulus $N = p_1 \ldots p_s$ and an "outer" modulus $N_{\text{outer}} = P_1 \ldots, P_s$ (for $s = \text{poly}(\lambda, \sum_i u_i)$), where $p_1, \ldots, p_s, P_1, \ldots, P_s$ are primes of bit-length $\text{poly}(\lambda, \sum_i u_i)$, and each P_i is much larger than p_i.

Now, suppose we want to encode a value $m \in \mathbb{Z}_N$, and define $m_i = m \bmod p_i$ for each $i \in [s]$. Under the CLT scheme, an encoding of m at an index set $T = A_1^{t_1} \ldots A_\ell^{t_\ell}$ is the value $c \in \mathbb{Z}_{N_{\text{outer}}}$ such that

$$c \equiv \frac{m_i + r_i p_i}{z_1^{t_1} \cdots z_\ell^{t_\ell}} \pmod{P_i}$$

for each P_i, where each r_i (the "noise" term) is a fresh random integer much smaller than P_i, and each of $z_1, \ldots, z_\ell$ is uniform in $\mathbb{Z}_{P_i}$ (generated once during $\mathsf{CM.Setup}$, and reused for each encoding). Addition and multiplication in the multilinear map, under the CLT scheme, now just correspond to addition and multiplication modulo N_{outer}. It is easy to verify that for addition of two encoded values $m, m' \in \mathbb{Z}_N$ within the same index set $T = A_1^{t_1} \ldots A_\ell^{t_\ell}$, we have

$$\frac{m_i + r_i p_i}{z_1^{t_1} \cdots z_\ell^{t_\ell}} + \frac{m'_i + r'_i p_i}{z_1^{t_1} \cdots z_\ell^{t_\ell}} = \frac{(m_i + m'_i) + (r_i + r'_i) p_i}{z_1^{t_1} \cdots z_\ell^{t_\ell}} \pmod{P_i}$$

where the new "noise" parameter, $r_i + r'_i$, has grown somewhat compared to r_i, r'_i (as described in Section 2.7), but for a bounded number of operations remains very small relative to P_i. Multiplication between encoded values $m, m' \in \mathbb{Z}_N$ at different index sets (resp, $T = A_1^{t_1} \ldots A_\ell^{t_\ell} \subset \mathcal{U}$ and $W = A_1^{w_1} \ldots A_\ell^{w_\ell} \subset \mathcal{U}$) operates similarly, with the result

$$\frac{m_i + r_i p_i}{z_1^{t_1} \cdots z_\ell^{t_\ell}} \cdot \frac{m'_i + r'_i p_i}{z_1^{w_1} \cdots z_\ell^{w_\ell}} = \frac{m_i m'_i + (r_i m'_i + r'_i m_i + r_i r'_i p_i) p_i}{z_1^{t_1 + w_1} \cdots z_\ell^{t_\ell + w_\ell}} \pmod{P_i}$$

now encoded at the product index set $TW = A_1^{t_1 + w_1} \ldots A_\ell^{t_\ell + w_\ell} \subset \mathcal{U}$, and with noise growth $(r_i m'_i + r'_i m_i + r_i r'_i p_i)$: faster than that of additions, but still much smaller than P_i for a bounded number of operations.

[11] More precisely, fix an arithmetic expression C of depth d and total degree r, and suppose we evaluate C on freshly encoded ring elements. The number of monomials in the expansion of C is at most 2^{dr}, so the noise bound of the resulting term is at most $2^{dr} \cdot (2^{f(\lambda)})^r$, and we will remain under the noise limit as long as $(d + f(\lambda))r < g(\lambda)$. In most cases of interest, we have $d \ll r$—in fact, if a constant fraction of the layers of C consist of multiplication gates, then $d = O(\lg r)$—and thus we can approximate the noise bound simply in terms of the degree.

In order to enable zero-testing, the CLT scheme also includes elements of the following form in the public parameters, for random integers h_i much smaller than P_i.

$$p_{\text{zt}} = \sum_{i \in [s]} \left(\frac{h_i \cdot z_1^{u_1} \cdots z_\ell^{u_\ell}}{p_i} \bmod P_i \right) \prod_{j \neq i \in [s]} P_j \pmod{N_{\text{outer}}}$$

Suppose we would like to zero-test an encoding $c = [m]_{\mathcal{U}} \in \mathbb{Z}_{N_{\text{outer}}}$, encoded at the top-level index set $\mathcal{U}$, so that $(t_1, \ldots, t_\ell) = (u_1, \ldots, u_\ell)$. Under the CLT scheme, this is done by computing $d = c \cdot p_{\text{zt}} \in \mathbb{Z}_{N_{\text{outer}}}$, and returning "zero" if the result is very small relative to N_{outer}, and "nonzero" otherwise.[12] Indeed, we can verify that if c encodes zero (i.e., $m = 0 \in \mathbb{Z}_N$), then every component $m_i = m \bmod p_i$ is zero in $\mathbb{Z}_{p_i}$, and we have

$$c \cdot p_{\text{zt}} \;=\; \sum_{i \in [s]} \left(\frac{r_i p_i}{z_1^{u_1} \cdots z_\ell^{u_\ell}} \cdot \frac{h_i \cdot z_1^{u_1} \cdots z_\ell^{u_\ell}}{p_i} \bmod P_i \right) \cdot \prod_{j \neq i} P_j \;=\; \sum_{i \in [s]} r_i h_i \prod_{j \neq i} P_j$$

and, since each r_i, h_i is much smaller than P_i, the result $c \cdot p_{\text{zt}} \in \mathbb{Z}_{N_{\text{outer}}}$ is very small (as an integer) relative to N_{outer}. On the other hand, if c does *not* encode zero (i.e., $c = [m]_{\mathcal{U}}$ for $m \neq 0 \in \mathbb{Z}_{N_{\text{outer}}}$), then at least one component m_i is nonzero in $\mathbb{Z}_{p_i}$; this introduces the extra additive term $(m_i h_i / p_i \bmod P_i) \prod_{j \neq i} P_j$, which with high probability is not very small relative to N_{outer}. Likewise, if T is *not* the top-level index set $\mathcal{U}$, then this introduces the factor $z_1^{u_1 - t_1} \cdots z_\ell^{u_\ell - t_\ell}$ modulo every P_i, and this again makes the result (with high probability) not small relative to N_{outer}. For more details, we refer the reader to the original work [CLT13].

In order to use the CLT scheme as a composite-order multilinear map with inner modulus $N = N_1 \cdots N_k$, setting the parameters requires some care, since the scheme must remain secure even when the adversary sees encodings that are zero in one or more of the subrings $(\mathbb{Z}_{N_1}, \ldots, \mathbb{Z}_{N_k})$. Gentry et al. [GLW14] investigate this question, and conclude that if each modulus $N_1, \ldots, N_k$ is a product of many (i.e., $\text{poly}(\lambda)$) of the primes among $p_1, \ldots, p_s$, then the scheme resists obvious attacks. This is the parameter setting that we adopt in this work. For the full analysis, we refer the reader to [GLW14, Appendix B].

Possible approaches for clean maps. While we know how to instantiate "noisy" multilinear maps via the CLT scheme, instantiation of "clean" multilinear maps remains a central open problem. Current techniques for "noisy" maps [GGH13a, CLT13, GGH14] depend crucially on the noise to hide the encoded elements. Even if it is possible to extend these techniques, and thereby reduce the noise below quadratic in the multilinearity degree, it seems very unlikely that the noise can be made only *polylogarithmic* in the degree. However, randomized encodings are not the only possible route to "clean" multilinear maps. The theory of bilinear maps [Mil04, MOV93, Jou00, BF01] does not incorporate noise at all, but rather relies on algebraic properties of pairings over elliptic curves. We believe that the most promising route to constructing clean multilinear maps is via structures that generalize these properties, such as abelian varieties. Some conditional negative results were presented by Boneh and Silverberg [BS03], but in general, the problem remains wide open.

2.9 Program Obfuscation

Our definition of VBB obfuscation is similar to the one studied in [BGK+14]. It is stronger than the original definition [BGI+01], in that we allow the adversary to output a string of arbitrary length,

[12] In fact, the CLT scheme uses multiple (polynomially many) such parameters p_{zt}, and the output is "zero" if the result $c \cdot p_{\text{zt}}$ is small with respect to all of them.

rather than just a single bit. In addition, the definition is parameterized over an ideal functionality (represented by a stateful oracle $\mathcal{M}$), to which both the obfuscator and the adversary have access. If $\mathcal{M}$ were the empty oracle, we would recover the usual definition of (strong) VBB obfuscation. In our setting, however, as in that of [BGK+14], the oracle $\mathcal{M}$ corresponds to our generic model of composite-order multilinear maps (Definition 2.10).

Definition 2.14 (Virtual Black-Box Obfuscation in an $\mathcal{M}$-Idealized Model ([BGI+01, BGK+14]))**.** Let $\mathcal{C} = (\mathcal{C}_\lambda)_{\lambda\in\mathbb{N}}$ be a family of Boolean circuits, and let $\mathcal{M}$ be a stateful oracle (possibly randomized). We say that a PPT machine $\mathcal{O}$ is a *virtual black-box obfuscator* for $\mathcal{C}$ in the $\mathcal{M}$-idealized model, if the following conditions are satisfied.

- Correctness: There is a negligible function ε such that for all $\lambda \in \mathbb{N}$, every circuit $C \in \mathcal{C}_\lambda$, every input $\mathbf{x}$ to C, and all possible random coins for $\mathcal{M}$, we have
$$\Pr[(\mathcal{O}^{\mathcal{M}}(1^\lambda, C))(\mathbf{x}) \neq C(\mathbf{x})] \ < \ \varepsilon(\lambda) \ .$$
- Virtual Black-Box: For every PPT adversary $\mathcal{A}$, there is a PPT simulator $\mathcal{S}$ such that for every PPT distinguisher D, there is a negligible function ε such that for all $C \in \mathcal{C}_\lambda$, we have
$$\left|\Pr[D(\mathcal{A}^{\mathcal{M}}(\mathcal{O}^{\mathcal{M}}(1^\lambda, C))) = 1] - \Pr[D(\mathcal{S}^C(1^\lambda, 1^{|C|})) = 1]\right| \ < \ \varepsilon(\lambda) \ ,$$
where the probability is over the coins of $D, \mathcal{A}, \mathcal{S}, \mathcal{O}$, and $\mathcal{M}$.

We note that since we require the obfuscator $\mathcal{O}$ to be efficient, the output of $\mathcal{O}$ is also a circuit of size $\text{poly}(\lambda)$ (and thus the *polynomial slowdown* property of Barak et al. [BGI+01] is immediate from the definition). We also extend Definition 2.14 in the standard way to entire circuit classes such as NC^1 and P/poly, as follows.

Definition 2.15 (Virtual Black-Box Obfuscation for P/poly)**.** Suppose that for every polynomial f, the PPT machine $\mathcal{O}$ is a virtual black-box obfuscator for the circuit family $(\mathcal{C}_\lambda)_{\lambda\in\mathbb{N}}$, where $\mathcal{C}_\lambda$ is the set of Boolean circuits $C : \{0,1\}^\lambda \to \{0,1\}$ of size at most $f(\lambda)$. Then $\mathcal{O}$ is a *virtual black-box obfuscator for P/poly* (in the $\mathcal{M}$-idealized model).

Definition 2.16 (Virtual Black-Box Obfuscation for NC^1)**.** Suppose that for every polynomial f and constant $c \in \mathbb{N}$, the PPT machine $\mathcal{O}$ is a virtual black-box obfuscator for the circuit family $(\mathcal{C}_\lambda)_{\lambda\in\mathbb{N}}$, where $\mathcal{C}_\lambda$ is the set of Boolean circuits $C : \{0,1\}^\lambda \to \{0,1\}$ of size at most $f(\lambda)$ and depth at most $c \lg \lambda$. Then $\mathcal{O}$ is a *virtual black-box obfuscator for* NC^1 (in the $\mathcal{M}$-idealized model).

2.10 Keyed and Succinct Obfuscation

As discussed in Section 2.4, the model of "keyed" programs is especially natural for program obfuscation. We now state a modified definition of VBB obfuscation, suited to this setting.

Definition 2.17 (Keyed Virtual Black-Box Obfuscation)**.** Fix a family of arithmetic circuits $\mathcal{C} = (C_\lambda)_{\lambda\in\mathbb{N}}$ (Section 2.3). For a stateful oracle $\mathcal{M}$ (possibly randomized), we say a pair of PPT algorithms ($\mathcal{O}$, $\mathcal{O}$.Eval) is a *keyed virtual black-box obfuscator* for $\mathcal{C}$ in the $\mathcal{M}$-idealized model, if the following conditions are satisfied.

- Correctness: There is a negligible function ε such that the following holds. Fix $\lambda \in \mathbb{N}$ and an arithmetic circuit $C \in \mathcal{C}_\lambda$, where $C : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$. Then for every input $\mathbf{x} \in \{0,1\}^n$ and key $\mathbf{y} \in \{0,1\}^m$, and all possible random coins for $\mathcal{M}$, we have
$$\Pr[\tilde{C}_\mathbf{y} \leftarrow \mathcal{O}^{\mathcal{M}}(C, \mathbf{y}) \ ; \ \mathcal{O}.\text{Eval}^{\mathcal{M}}(\tilde{C}_\mathbf{y}, C, \mathbf{x}) \neq C(\mathbf{x}, \mathbf{y})] \ < \ \varepsilon(\lambda) \ ,$$
where the probability is over the coins of $\mathcal{O}$.

- Virtual Black-Box: For every PPT adversary $\mathcal{A}$, there is a PPT simulator $\mathcal{S}$ such that for all PPT distinguishers D, and all $(C, n, m) \in \mathcal{C}$, we have

$$\left|\Pr[D(\mathcal{A}^{\mathcal{M}}(\mathcal{O}^{\mathcal{M}}(C, \mathbf{y})) = 1] - \Pr[D(\mathcal{S}^{C(\cdot,\mathbf{y})}(C)) = 1]\right| < \mathrm{negl}(|C|) \ ,$$

  where the probability is over the coins of $D, \mathcal{A}, \mathcal{S}, \mathcal{O}$, and $\mathcal{M}$.

Intuitively, the definition of *keyed* program obfuscation separates the question of the *public* ("universal") circuit parameters from the size of the secret part of the circuit, which is to be obfuscated. It now makes sense to discuss *succinct* program obfuscation, in which the obfuscation size is independent of the public part of the circuit, and depends only on the secret key (and on the security parameter).

Definition 2.18 (Succinct Virtual Black-Box Obfuscation)**.** The definition is the same as Definition 2.17, with the following additional requirement.

- Succinctness: There exists a polynomial f such that for all $(C, n, m) \in \mathcal{C}$ and all $\mathbf{y} \in \{0,1\}^m$, we have $|\mathcal{O}^{\mathcal{M}}(C, \mathbf{y})| \leq f(n, m, \lambda)$.

We also specialize Definition 2.18 to the classes P/poly and NC^1, just as in Definitions 2.15 and 2.16. Since the details are identical, we omit the formal definitions.

2.11 Straddling Sets

Our obfuscator uses the multilinear map's index sets (Definition 2.7) to enforce constraints on the adversary's evaluation. This requires careful design of the indices for each element in the multilinear map. To simplify the presentation of our design, we now introduce some simple combinatorial properties that we use in our security proof.

One important building block is the notion of *straddling sets*, as described by Barak et al. [BGK+14]. Roughly speaking, an n-straddling set system consists of two partitions $\mathcal{S}_0, \mathcal{S}_1$ of the set $\{1, \ldots, n\}$, such that once we choose a set from (say) $\mathcal{S}_0$, we have committed to $\mathcal{S}_0$, and we cannot complete this set to form a full partition of $\{1, \ldots, n\}$ except by adding all (and only) the remaining sets in the partition $\mathcal{S}_0$. In fact, we require a slightly stronger property, which we now formalize.

Definition 2.19 (Partition)**.** For a set $\mathcal{S}$ and an integer n, we say $(S_1, \ldots, S_n)$ is a *partition* of $\mathcal{S}$ if each S_i is a nonempty set, $S_1 \cup \ldots \cup S_n = \mathcal{S}$, and for each $i \neq j \in [n]$, we have $S_i \cap S_j = \emptyset$.

Definition 2.20 (Straddling Set Systems ([BGK+14], adapted))**.** For $n \in \mathbb{N}$, an *n-straddling set system over a set $\mathcal{S}$* consists of two partitions of $\mathcal{S}$:

$$\mathcal{S}_0 = (S_{0,1}, \ldots, S_{0,n}) \ , \quad \mathcal{S}_1 = (S_{1,1}, \ldots, S_{1,n})$$

with the following property. Fix $\mathcal{T} \subseteq \mathcal{S}$, and let T_0, T_1 be subsequences of $S_{0,1}, \ldots, S_{0,n}, S_{1,1}, \ldots, S_{1,n}$ such that each of T_0, T_1 is a partition of $\mathcal{T}$, and $T_0 \neq T_1$ (i.e., they are not the same subsequence). Then each of T_0, T_1 is one of the original partitions $\mathcal{S}_0, \mathcal{S}_1$, and $\mathcal{T} = \mathcal{S}$.

As an immediate corollary, we note that the only partitions of $\mathcal{S}$ that can be formed using sets in $\mathcal{S}_0, \mathcal{S}_1$ are the partitions $\mathcal{S}_0, \mathcal{S}_1$ themselves.

Corollary 2.21. *Fix an integer n, a set $\mathcal{S}$, and an n-straddling set system $(\mathcal{S}_0 = (S_{0,1}, \ldots, S_{0,n}),\ \mathcal{S}_1 = (S_{1,1}, \ldots, S_{1,n}))$ over $\mathcal{S}$. Let T be a subsequence of $S_{0,1}, \ldots, S_{0,n}, S_{1,1}, \ldots, S_{1,n}$. If T is a partition of $\mathcal{S}$, then either $T = \mathcal{S}_0$ or $T = \mathcal{S}_1$.*

Proof. Suppose $T \neq \mathcal{S}_1$. Then T and $\mathcal{S}_1$ each partition $\mathcal{S}$ but are not the same subsequence, and by Definition 2.20 we conclude $T = \mathcal{S}_0$. □

In fact, the simple construction of straddling sets in [BGK+14] already satisfies our stronger definition. For completeness, we reproduce their construction and extend the proof to our setting.

Construction 2.22 (Construction of Straddling Set Systems ([BGK+14], adapted))**.** Fix an integer n. We construct an n-straddling set system over the set $\mathcal{S} = [2n-1]$ (and hence over any set with $2n-1$ elements). Define:

$$\mathcal{S}_0 = (S_{0,1}, \ldots, S_{0,n}) = (\{1\}, \{2,3\}, \{4,5\}, \ldots, \{2n-4, 2n-3\}, \{2n-2, 2n-1\})$$

$$\mathcal{S}_1 = (S_{1,1}, \ldots, S_{1,n}) = (\{1,2\}, \{3,4\}, \{5,6\}, \ldots, \{2n-3, 2n-2\}, \{2n-1\})$$

Lemma 2.23 (Straddling Set Systems ([BGK+14], adapted))**.** *For each integer n, Construction 2.22 is an n-straddling set system over $[2n-1]$.*

Proof. Let T_0, T_1 be subsequences of $S_{0,1}, \ldots, S_{0,n}, S_{1,1}, \ldots, S_{1,n}$, such that each of T_0, T_1 is a partition of the same subset $\mathcal{T} \subseteq [2n-1]$. We proceed by induction on the size of $\mathcal{T}$. Consider any maximal contiguous interval $[a,b]$ in $\mathcal{T}$ (i.e., so that $[a,b] \subseteq \mathcal{T}$ but $a-1, b+1 \notin \mathcal{T}$). If a is even, then the only set containing a but not $a-1$ is $S_{0,a/2}$, so this set must be in both T_0 and T_1; removing it reduces the size of $\mathcal{T}$ by 2 and we invoke the inductive hypothesis. If a is odd but not equal to 1, then the same argument applies with $S_{1,(a+1)/2}$; if b is even, the same argument applies with $S_{1,b/2}$; and if b is odd but not equal to $2n-1$, then the same argument applies with $S_{0,(b+1)/2}$. So the only remaining case is $a = 1$ and $b = 2n-1$, and hence $\mathcal{T} = [2n-1]$.

Now, since T_0 covers the element $a = 1$, it must contain either $S_{0,1}$ or $S_{1,1}$. Without loss of generality suppose it contains $S_{0,1}$. Then since $\mathcal{S}_0$ also contains $S_{0,1}$, we find that $T_0 \setminus \{S_{0,1}\}$ and $\mathcal{S}_0 \setminus \{S_{0,1}\}$ are partitions that cover $[2, 2n-1]$, and by the inductive hypothesis we conclude $T_0 = \mathcal{S}_0$. Since the same argument applies to T_1, this proves the claim. □

3 Construction

We now present our main obfuscation construction (Construction 3.1), which acts on keyed circuits (Section 2.4) as depicted in Figure 3. (We note that we can obtain keyed circuit families from various other machine models, including general Boolean circuits, by the transformations of Section 2.4.)

Construction 3.1 (Virtual Black-Box Obfuscation)**.** Let $\mathsf{CM} = (\mathsf{CM.Setup}, \mathsf{CM.Add}, \mathsf{CM.Mult}, \mathsf{CM.ZeroTest}, \mathsf{CM.Encode})$ be a composite-order multilinear map (Definition 2.8). Fix an input $(C, \mathbf{y})$, where $\mathbf{y} \in \{0,1\}^m$, and $C : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$ is an arithmetic circuit (representing the keyed circuit $C_{\mathbf{y}}$, as in Section 2.4). Let d be the depth of the circuit C; let $\deg(\mathbf{y})$ be the total degree of C in all of the variables $y_1, \ldots, y_m$; and for each $i \in [n]$ let $\deg(x_i)$ be the degree of C in the variable x_i. For a security parameter $\lambda \in \mathbb{N}$ (represented in unary), the obfuscation procedure $\mathcal{O}(1^\lambda, C, \mathbf{y})$ operates as follows.

$\underline{\mathcal{O}(1^\lambda, C, \mathbf{y})}$:

1. For each $i \in [n]$, let $(S_{i,b,1}, \ldots, S_{i,b,n})_{b \in \{0,1\}}$ be an n-straddling set system (Lemma 2.23) over a set $\mathcal{S}_i$ of $(2n-1)$ fresh formal symbols. For each $b \in \{0,1\}$ and $i \in [n]$, define $\mathsf{BitCommit}_{i,b} = S_{i,b,i}$. For each $b_1, b_2 \in \{0,1\}$ and $i_1, i_2 \in [n]$ such that $i_1 < i_2$, define $\mathsf{BitFill}_{i_1,i_2,b_1,b_2} = S_{i_1,b_1,i_2} S_{i_2,b_2,i_1}$.

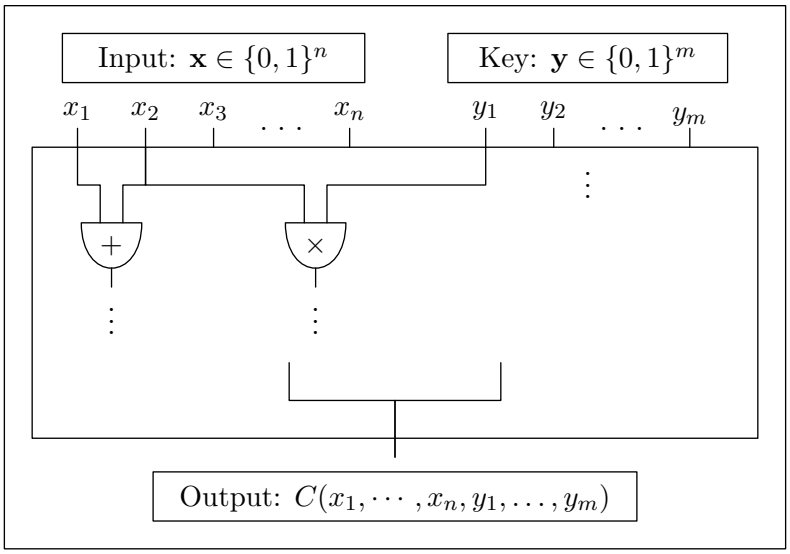

Figure 2: A keyed circuit family, defined by a "universal" arithmetic circuit $C : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$, which can be input to our main obfuscation construction (Construction 3.1). In Figure 3, below, we illustrate the evaluation procedure $\mathcal{O}.\mathsf{Eval}$ for this circuit C.

2. Construct the following index set of fresh formal symbols (Definition 2.7) as the top-level index set:
$$\mathcal{U} = Y^{\deg(\mathbf{y})} \prod_{i \in [n]} (X_{i,0} X_{i,1})^{\deg(x_i)} Z_i W_i \mathcal{S}_i$$

3. Run $(\mathsf{CM.pp}, \mathsf{CM.sp}, N_{\mathrm{ev}}, N_{\mathrm{chk}}) \leftarrow \mathsf{CM.Setup}(\mathcal{U}, 1^{d+\lambda}, 2)$, indicating a security parameter of $d+\lambda$ for the multilinear map, and a modulus that decomposes into two factors $N = N_{\mathrm{ev}} N_{\mathrm{chk}}$.

4. For each $i \in [n]$, generate uniformly random values $\alpha_i, \gamma_{i,0}, \gamma_{i,1} \leftarrow \mathbb{Z}^*_{N_{\mathrm{chk}}}$ and $\delta_{i,0}, \delta_{i,1} \leftarrow \mathbb{Z}^*_{N_{\mathrm{ev}}}$. For each $j \in [m]$, generate a uniformly random value $\beta_j \leftarrow \mathbb{Z}^*_{N_{\mathrm{chk}}}$.

5. Compute the check value $C^* = C(\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_m) \in \mathbb{Z}_{N_{\mathrm{chk}}}$.

6. Using $\mathsf{CM.Encode}(\mathsf{CM.sp}, \cdot)$, for $i \in [n]$, $j \in [m]$, and $b \in \{0,1\}$, generate the following encoded ring elements (using the notation of Remark 2.9):
$$\hat{x}_{i,b} = [b, \alpha_i]_{X_{i,b}} \qquad \hat{u}_{i,b} = [1,1]_{X_{i,b}} \qquad \hat{y}_j = [y_j, \beta_j]_Y \qquad \hat{v} = [1,1]_Y$$
$$\hat{z}_{i,b} = [\delta_{i,b}, \gamma_{i,b}]_{X_{i,1-b}^{\deg(x_i)} Z_i W_i \, \mathsf{BitCommit}_{i,b}} \qquad \hat{w}_{i,b} = [0, \gamma_{i,b}]_{W_i \, \mathsf{BitCommit}_{i,b}}$$
$$\hat{C}^* = [0,\ C^*]_{Y^{\deg(\mathbf{y})} \prod_{i \in [n]} (X_{i,0} X_{i,1})^{\deg(x_i)} Z_i}$$
For $b_1, b_2 \in \{0,1\}$ and each $i_1, i_2 \in [n]$ such that $i_1 < i_2$, generate the following encoded ring elements (using the notation of Remark 2.9):
$$\hat{s}_{i_1,i_2,b_1,b_2} = [1,1]_{\mathsf{BitFill}_{i_1,i_2,b_1,b_2}}$$
For notational convenience, for each $i_2 < i_1 \in [n]$, we also define $\hat{s}_{i_2,i_1,b_2,b_1} = \hat{s}_{i_1,i_2,b_1,b_2}$. We refer to the elements $\hat{u}_{i,b}, \hat{v}, \hat{s}_{i_1,i_2,b_1,b_2}$ as *unit encodings*, since they each encode $1 \in \mathbb{Z}_N$, and they are incorporated solely for their effect on the index sets.

7. Output the values above, along with the public parameters of the multilinear map:

$$\mathcal{O}(1^\lambda, C, \mathbf{y}) \;=\; \Big(\; \mathsf{CM.pp},\;\; (\hat{x}_{i,b}, \hat{u}_{i,b}, \hat{z}_{i,b}, \hat{w}_{i,b})_{i\in[n],b\in\{0,1\}},\;\; (\hat{y}_j)_{j\in[m]},\;\; \hat{v},\;\; \hat{C}^*,\; (\hat{s}_{i_1,i_2,b_1,b_2})_{b_1,b_2\in\{0,1\},i_1<i_2\in[n]} \;\Big)$$

To evaluate the obfuscated program $\tilde{C}_{\mathbf{y}} = \mathcal{O}(1^\lambda, C, \mathbf{y})$ on an input $\mathbf{x} = x_1 \cdots x_n \in \{0,1\}^n$, the evaluation procedure $\mathcal{O}.\mathrm{Eval}(\tilde{C}_{\mathbf{y}}, C, \mathbf{x})$ operates as follows.

$\mathcal{O}.\mathrm{Eval}(\tilde{C}_{\mathbf{y}}, C, \mathbf{x})$:

1. Using the procedures $\mathsf{CM.Add}(\mathsf{CM.pp}, \cdot, \cdot)$ and $\mathsf{CM.Mult}(\mathsf{CM.pp}, \cdot, \cdot)$, along with the unit encodings $(\hat{u}_{i,x_i}, \hat{v})$, evaluate the circuit C on the encoded inputs $\hat{x}_{1,x_1}, \ldots, \hat{x}_{n,x_n}, \hat{y}_1, \ldots, \hat{y}_m$. In other words, substitute the values $\hat{x}_{1,x_1}, \ldots, \hat{y}_m$ for the corresponding input wires $x_{1,x_1}, \ldots, y_m$; and, for each gate in the circuit, substitute one of the following operations:

   - For a multiplication gate, on operands $[a]_S, [b]_T$, output $\mathsf{CM.Mult}(\mathsf{CM.pp}, [a]_S, [b]_T) = [ab]_{ST}$.
   - For an addition gate, we cannot substitute an invocation of $\mathsf{CM.Add}$ (since the index sets of the encoded operands need not match), so instead we substitute the following procedure (Figure 3, box "$\mathcal{O}(+)$"). Suppose the input values to the addition gate are the encoded elements $[a]_S, [b]_T$ for index sets $S, T \subseteq \mathcal{U}$. Using $\mathsf{CM.Mult}$, multiply each term $[a]_S, [b]_T$ by the powers of unit encodings $(\hat{u}_{i,x_i}, \hat{v})$ that are minimally necessary to make the index set $S \cup T$ for both resulting elements. Then, using $\mathsf{CM.Add}$, output the sum of the two.

   We note that the result of this procedure, for each sub-circuit of C, will be an encoding whose index set contains factors corresponding to each input variable ($X_{i,b}, Y$, resp., for $\hat{x}_{i,b}, \hat{y}_j$), raised to the power of the degree of the given sub-circuit in those variables. Thus in particular, at the end of the evaluation, the final term will be encoded at the index set $Y^{\deg(\mathbf{y})} \prod_{i\in[n]} X_{i,x_i}^{\deg(x_i)}$. We denote this final term $\hat{C}$ as follows:

$$\hat{C} = [C(x_1, \ldots, x_n, y_1, \ldots, y_m), C(\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_m)]_{Y^{\deg(\mathbf{y})} \prod_{i\in[n]} X_{i,x_i}^{\deg(x_i)}}$$

   (We remark that while we present simple algorithms here for clarity, there are many natural optimizations; for details, we refer the reader to Section 4.)

2. Using the procedures $\mathsf{CM.Add}$, $\mathsf{CM.Mult}$, compute the following unit encoding:

$$\hat{\sigma} = \prod_{i_1<i_2\in[n]} \hat{s}_{i_1,i_2,x_{i_1},x_{i_2}}$$

3. Using the procedures $\mathsf{CM.Add}$, $\mathsf{CM.Mult}$, compute the following encoded element:

$$z \;=\; \left(\hat{C} \prod_{i\in[n]} \hat{z}_{i,x_i} \;-\; \hat{C}^* \prod_{i\in[n]} \hat{w}_{i,x_i} \right) \cdot \hat{\sigma}$$

4. Run $\mathsf{CM.ZeroTest}(\mathsf{CM.pp}, z)$. If it outputs "zero", output 0; if "nonzero", output 1.

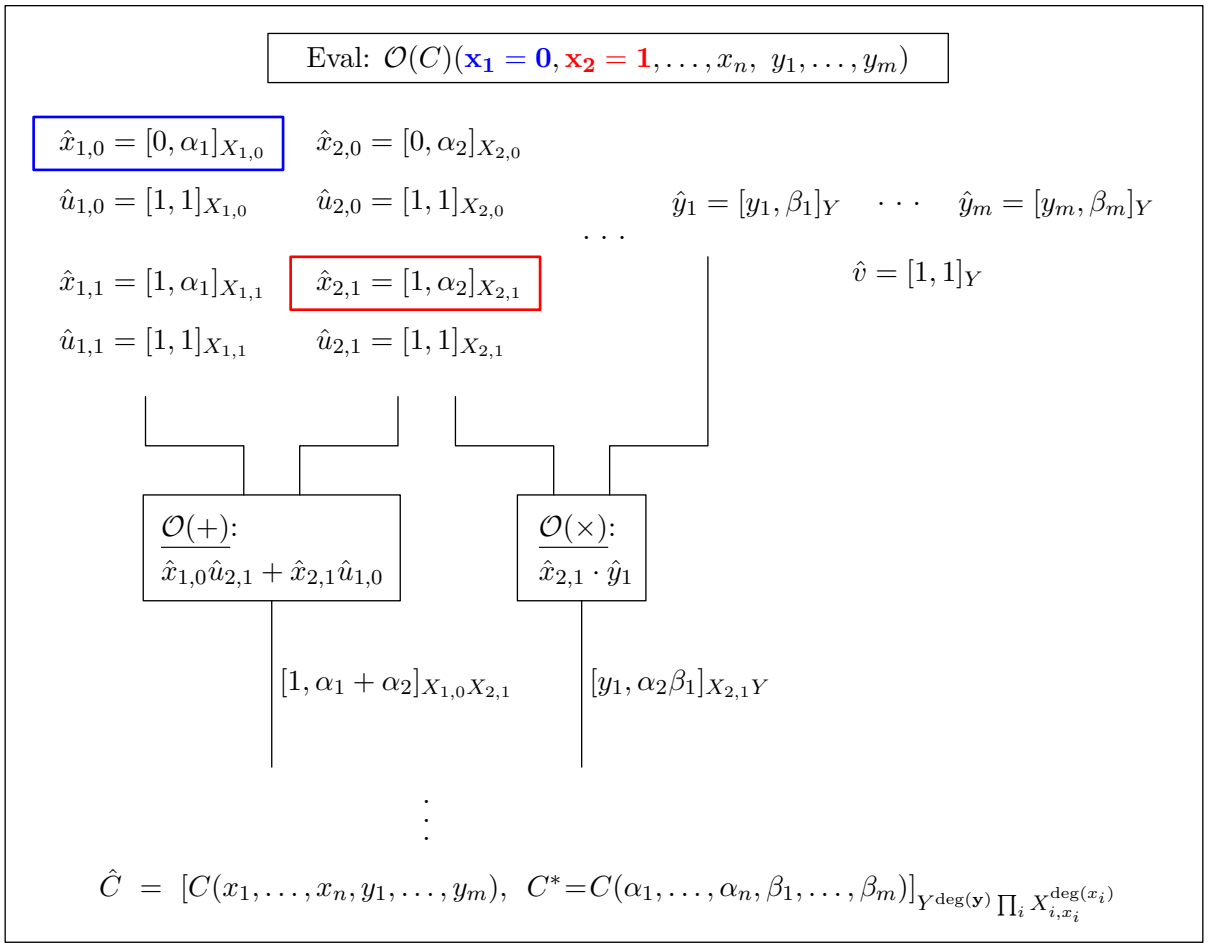

Figure 3: The first step of the evaluation procedure $\mathcal{O}$.Eval, for an obfuscated version of the keyed arithmetic circuit C from Figure 2. First, we use the bits of the input string $\mathbf{x}$ (e.g., $x_1 = 1$, $x_2 = 0$, ..., x_n) to select the relevant input encodings $\hat{x}_{1,1}, \hat{x}_{2,0}, \ldots, \hat{x}_n$. We then run C directly on the encodings $\hat{x}_{1,1}, \hat{x}_{2,0}, \ldots, \hat{y}_1, \ldots, \hat{y}_m$, substituting multilinear map operations for the corresponding arithmetic gates, and using the unit encodings $\hat{u}_{i,x_i}, \hat{v}$ to raise the index sets of intermediate encodings as needed so that the operands to addition gates match.

Correctness of the construction. We first show that Construction 3.1 is correct, which is fairly straightforward from the definitions of the multilinear map operations.

Lemma 3.2 (Correctness of Construction 3.1)**.** *Let the procedures $\mathcal{O}, \mathcal{O}$.Eval be defined as in Construction 3.1, and fix an arithmetic circuit $C : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$, a key $\mathbf{y} \in \{0,1\}^m$, and an input $\mathbf{x} \in \{0,1\}^n$. If $\tilde{C}_\mathbf{y} \leftarrow \mathcal{O}(C, \mathbf{y})$, then we have $\mathcal{O}.\mathrm{Eval}(\tilde{C}_\mathbf{y}, C, \mathbf{x}) = C(\mathbf{x}, \mathbf{y})$.*

Proof. First, we show that the zero-test query z has the correct index set. As described above, the result of the evaluation, $\hat{C}$, has the index set $Y^{\deg(\mathbf{y})} \prod_i X_{i,x_i}^{\deg(x_i)}$, while each $\hat{z}_{i,x_i}$ has the index set $X_{i,1-x_i}^{\deg(x_i)} Z_i W_i \mathsf{BitCommit}_{i,x_i}$. The variable $\hat{C}^*$ has the index set $Y^{\deg(\mathbf{y})} \prod_i (X_{i,0}X_{i,1})^{\deg(x_i)} Z_i$, while each $\hat{w}_{i,x_i}$ has the index set $W_i \mathsf{BitCommit}_{i,x_i}$. Thus, both terms $\hat{C} \prod_i \hat{z}_{i,x_i}$ and $\hat{C}^* \prod_i \hat{w}_{i,x_i}$ have the following index set:

$$Y^{\deg(\mathbf{y})} \prod_{i \in [n]} (X_{i,0}X_{i,1})^{\deg(x_i)} Z_i W_i S_{i,x_i,i}$$

Finally, each variable $\hat{s}_{i_1,i_2,x_{i_1},x_{i_2}}$ has the index set $\mathsf{BitFill}_{i_1,i_2,x_{i_1},x_{i_2}} = S_{i_1,x_{i_1},i_2} S_{i_2,x_{i_2},i_1}$, and thus

the index set of the entire query z is the top-level index set $\mathcal{U}$, as desired.

Now, we consider the value of each zero-test query z, in both components (modulo N_{ev} and modulo N_{chk}). Since the unit encodings $\hat{s}_{i_1,i_2,x_{i_1},x_{i_2}}$ take the value $1 \in \mathbb{Z}_N$, they do not affect the real value of z, so we can drop the factor of $\hat{\sigma}$, and we need only consider the value of the following expression:

$$z' = \hat{C} \prod_{i \in [n]} \hat{z}_{i,x_i} - \hat{C}^* \prod_{i \in [n]} \hat{w}_{i,x_i}$$

The first component of this value z' will evaluate to $C(\mathbf{x}, \mathbf{y}) \cdot \prod_i \delta_{i,x_i}$, and since each δ_{i,x_i} is invertible in $\mathbb{Z}_{N_{\text{ev}}}$, this quantity is zero precisely when $C(\mathbf{x}, \mathbf{y}) = 0$. Meanwhile, the second component will be $C(\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_m) \cdot \prod_i \gamma_{i,x_i} - C(\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_m) \cdot \prod_i \gamma_{i,x_i} = 0$ always. □

Succinctness. In Construction 3.1, we instantiate the multilinear map with a security parameter of $d + \lambda$, rather than λ. As we will see in Section 3.5, this term reflects the bound from the Schwartz-Zippel identity testing algorithm. This is somewhat unsatisfying, since it prevents us from constructing *succinct* obfuscation (Definition 2.18), and intuitively it does not seem necessary to prove security. Indeed, it turns out that if we assume the hardness of factoring, then we can eliminate the extra term. We will explore this adaptation and its proof in Section 3.5; for now, we just state the modified ("succinct") version of the construction.

Construction 3.3 (Virtual Black-Box Obfuscation (Succinct Version))**.** Proceed as in Construction 3.1, except in step 3, provide 1^λ as the security parameter to $\mathsf{CM.Setup}$, rather than $1^{d+\lambda}$.

Remark 3.4 (Indistinguishability Obfuscation)**.** Our main results will show that Construction 3.1 achieves VBB obfuscation in the generic model of composite-order multilinear maps. However, we note that if we only need the weaker notion of *indistinguishability obfuscation* [BGI+01], then we can obtain better parameters by eliminating some of the encodings. For continuity, we defer the details of this modification to Appendix A.

3.1 Main Theorems

We now state our main theorems, which show that our construction achieves VBB obfuscation in a generic model of composite-order multilinear maps. Our construction works either with "noisy" multilinear maps, which can be instantiated with the CLT scheme, or with "clean" maps, whose existence is still open. Since we work with circuits directly, unlike previous approaches which first convert them to branching programs, there is no *inherent* reason that our construction cannot be applied directly to all polynomial-size circuits.

Indeed, assuming "clean" maps (whose existence is still open), we are able to prove VBB obfuscation for P/poly (in the generic model) directly, without the additional assumption of FHE as in the work of Garg et al. [GGH+13b]. In fact, under the additional assumption that factoring integers is hard on average, we are also able to show that our construction (in its *succinct* variant, Construction 3.3) achieves *succinct* VBB obfuscation (Definition 2.18) for P/poly.

Theorem 3.5. *Suppose that factoring is hard (Assumption 3.11). Then Construction 3.3 achieves* succinct *virtual black-box obfuscation for P/poly in the generic model of* clean *composite-order multilinear maps.*

For completeness, we also prove the non-succinct version of Theorem 3.5, since there we do not assume the hardness of factoring.[13]

Theorem 3.6. *Construction 3.1 (composed with a universal circuit simulation, Construction 2.4) achieves virtual black-box obfuscation for P/poly in the generic model of* clean *composite-order multilinear maps.*

Of course, it is still unknown whether "clean" multilinear maps exist, and thus we prove separately that we achieve obfuscation for NC^1 given only "noisy" maps, which can be instantiated with the CLT scheme. As usual, we are unable to construct obfuscation for poly-size circuits directly from "noisy" maps, since the noise growth still increases with the degree (which is potentially exponential in the circuit depth). Still, we note that our construction is somewhat more general than the theorem suggests: even with "noisy" maps, our construction also works for *arithmetic* circuits whose depth is superlogarithmic but whose degree remains polynomial.

Theorem 3.7. *Construction 3.1 (composed with a universal circuit simulation, Construction 2.5) achieves virtual black-box obfuscation for* NC^1 *in the generic model of* noisy *composite-order multilinear maps.*

In the "noisy" case, we do not prove the corresponding theorem for *succinct* obfuscation, since in our definition (and in all known instantiations), the representation size of a ring element in a "noisy" multilinear map grows with the degree of multilinearity required. However, we remark that the analogous theorem would hold in the case of "noisy" multilinear maps whose representation size was nevertheless independent of the noise bound—the existence of such maps is also unknown.

In order to prove our main theorems (Theorems 3.5, 3.6, and 3.7), we first introduce a number of key ingredients in the proof.

3.2 Proof Ingredient: The Schwartz-Zippel Algorithm

In our obfuscation construction, the values in the second component of the direct product ($\mathbb{Z}_{N_{\mathrm{chk}}}$) force the adversary to run the Schwartz-Zippel algorithm implicitly on his chosen query, making the result useless if he submits the wrong polynomial. In order to use this fact in our security proof, we first recall the Schwartz-Zippel lemma and the corresponding algorithm for polynomial identity testing. We adapt the algorithm to our setting, in which we work with polynomials over the integers, but must test them using finite fields $\mathbb{Z}_p$ for (independent) random primes p.

Construction 3.8 (Schwartz-Zippel Algorithm ([DL78, Zip79, Sch80], adapted))**.** We describe an efficient randomized algorithm SZTest to decide whether a given arithmetic circuit $C(x_1, \ldots, x_n)$ computes a multivariate polynomial that is identically zero in $\mathbb{Z}[x_1, \ldots, x_n]$. The algorithm takes as input the circuit C, along with a security parameter λ (written in unary).

$\underline{\mathsf{SZTest}(1^\lambda, C)}$:

1. Choose a prime $p \leftarrow \mathsf{Primes}[2^\lambda, 2^{\lambda+1}]$, and values $\alpha_1, \ldots, \alpha_n \leftarrow \mathsf{Uniform}(\mathbb{Z}_p^*)$.
2. Output C

We note that step (2) is polynomial-time by construction, while the prime number theorem, along with any efficient primality test, implies that step (1) is also polynomial-time. Further, if $C \equiv 0$ then by definition SZTest outputs "zero". So the only interesting case is $C \not\equiv 0$, for which we now state an adaptation of the original Schwartz-Zippel lemma.

Lemma 3.9 (Schwartz-Zippel Lemma ([DL78, Zip79, Sch80], adapted))**.** *Let p be a prime, and let $f \in \mathbb{Z}_p[x_1, \ldots, x_n]$ be a multivariate polynomial of total degree δ, not identically zero. Then we have the following bound:*

$$\Pr[\, \alpha_1, \ldots, \alpha_n \leftarrow \mathsf{Uniform}(\mathbb{Z}_p) \;;\; f(\alpha_1, \ldots, \alpha_n) = 0 \,] < \delta/p$$

In our main proofs below, we will want to apply the Schwartz-Zippel lemma not just over prime-order finite fields, but also over the integers. We state the following corollary, which shows that for polynomials in $\mathbb{Z}[x_1, \ldots, x_n]$, an analogous result holds, as long as the coefficient of each monomial is bounded appropriately (e.g., due to the fact that the polynomial is computed by a small circuit).

Corollary 3.10. *For all polynomials r, s, there is a negligible function ε such that the following holds. For all integers $n, \lambda > 0$ and every arithmetic circuit $C \not\equiv 0 \in \mathbb{Z}[x_1, \ldots, x_n]$ of size at most $s = s(\lambda)$ and total degree at most $2^r = 2^{r(\lambda)}$, we have* $\Pr\left[\, \mathsf{SZTest}(1^{\lambda + r(\lambda)}, C) \to \text{"zero"} \,\right] < \varepsilon(\lambda)$.

Proof. Fix such an arithmetic circuit $C \not\equiv 0 \in \mathbb{Z}[x_1, \ldots, x_n]$, and consider the formal expansion of C (as a polynomial over the integers) into monomials in its formal variables $x_1, \ldots, x_n$, *without* collecting like terms. The number of monomials is at most $2^{2^r s}$, and thus even after collecting like terms, the maximum magnitude of the coefficient of any monomial is bounded by $2^{2^r s}$. Since $C \not\equiv 0$, one such coefficient is nonzero; let $A x_1^{a_1} x_2^{a_2} \cdots x_n^{a_n}$ be the lexicographically first such monomial, so that $A \neq 0$ and $|A| < 2^{2^r s}$. Now, $|A|$ has at most $2^r s$ distinct prime factors, and in particular, when $p \leftarrow \mathsf{Primes}[2^{r+\lambda}, 2^{r+\lambda+1}]$, the probability that p divides A is at most $2^r s / \Theta(2^{r+\lambda}/(r+\lambda)) = \mathrm{negl}(\lambda)$, by the prime number theorem. In other words, with overwhelming probability, the formal polynomial C is not identically zero when regarded modulo p. Since its degree is at most 2^r, the claim now follows by the Schwartz-Zippel lemma (Lemma 3.9). □

3.3 Proof Ingredient: Computational Schwartz-Zippel

The Schwartz-Zippel lemma (along with our adaptation, Corollary 3.10) suffices for our main security argument, but requires SZTest to be invoked with a security parameter that grows with the depth of the circuit being tested. Since we are not the ones running SZTest—rather, we are forcing the adversary to run the algorithm implicitly, in the second component of our ring elements—this means that the size of the modulus N_{chk} must grow with the depth of the circuit to be obfuscated. Even with a "clean" multilinear map, the parameters of the map would depend on each specific circuit, and we would not be able to construct *succinct* obfuscation (Definition 2.18), nor would it make sense to analyze our construction in terms of the number of ring elements and operations required (Section 4).

To solve this problem, we refer to an elegant idea of Boneh and Lipton [BL97]. In that work the authors observe that, assuming factoring integers is hard on average, it must be the case that polynomials $f \in \mathbb{Z}[x]$ with "too many" roots modulo "too many" primes are difficult to evaluate (roughly speaking). Indeed, if there were an efficient way to evaluate such f, then one could also evaluate f modulo a given composite $N = pq$, and with nonnegligible probability obtain a value that is zero modulo one of $\{p, q\}$ but not the other (and hence factor N).

We view the contrapositive of the Boneh-Lipton result as a *computational* analog of the Schwartz-Zippel lemma. In other words, even though there are many polynomials f of degree $2^d \gg 2^\lambda$ for

which $\mathsf{SZTest}(1^\lambda, f)$ is often wrong, we conjecture that no such f can be computed by small circuits. More precisely, we now state our generalization of the Boneh-Lipton result as Lemma 3.12, based on the hardness of factoring (which we formalize in a standard way).

Assumption 3.11 (Factoring is Hard)**.** *For every polynomial s, there is a negligible function ε such that the following holds. For every family of circuits $\mathcal{C} = (C_1, C_2, \ldots)$, where each $C_\ell : \{0,1\}^\ell \to \{0,1\}^\ell$ is a circuit of size at most $s(\ell)$, and every $r \in \mathbb{Z}$, we have*

$$\Pr[\ p, q \leftarrow \mathsf{Primes}[r, 2r]\ ;\ C_{2(\lg r+1)}(pq) \in \{p, q\}\ ] < \varepsilon(\lg r)\ .$$

Lemma 3.12 (Computational Schwartz-Zippel Lemma)**.** *Suppose factoring is hard (Assumption 3.11). Then for every polynomial s, there is a negligible function ε such that for all $\lambda \in \mathbb{N}$ and every arithmetic circuit $C \not\equiv 0$ (over the integers) of size at most $s(\lambda)$, we have* $\Pr\left[\ \mathsf{SZTest}(1^\lambda, C) \to \text{“zero”}\ \right] < \varepsilon(\lambda)$.

Proof. The proof is similar to that of Boneh and Lipton [BL97]. For sake of contradiction, suppose there is a polynomial s, a constant $c > 0$, and an increasing sequence of integers $(\lambda_1, \lambda_2, \ldots)$, such that for each λ_i, there is a circuit C_i of size at most $s(\lambda_i)$ for which $\Pr\left[\ \mathsf{SZTest}(1^\lambda, C) \to \text{“zero”}\ \right] > 1/\lambda^c$. We now consider the following algorithm for factoring in P/poly.

$\underline{\mathsf{CompSZFactor}(N)}$:

1. For each i such that $\lambda_i < \lg N$:
   (a) Choose values $\alpha_1, \ldots, \alpha_n \leftarrow \mathbb{Z}_N \setminus \{0, 1\}$ uniformly at random. If one of these values α_j is not prime to N, immediately output $\gcd(\alpha_j, N)$.
   (b) Compute $z = C_i(\alpha_1, \ldots, \alpha_n) \bmod N$. If z is not prime to N, immediately output $\gcd(z, N)$.
2. Output 1 (indicating failure).

We now bound the probability that $\mathsf{CompSZFactor}$ fails to output a proper factor of N, when N is generated as specified in Assumption 3.11. To do this, we first define the following quantity (for an arithmetic circuit $C \in \mathbb{Z}[x_1, \ldots, x_n]$):

$$\mathsf{RootDensity}(C, r) = \Pr\Big[\ p \leftarrow \mathsf{Primes}[r, 2r]\ ;\ x_1, \ldots, x_n \leftarrow \mathsf{Uniform}(\mathbb{Z}_p^*)\ ;\ C(x_1, \ldots, x_n) \equiv 0 \pmod p\ \Big]$$

Now fix some $\lambda = \lambda_i \in (\lambda_1, \lambda_2, \ldots)$, and the corresponding circuit $C = C_i$. By assumption, we have $\mathsf{RootDensity}(C, 2^\lambda) > \lambda^{-c}$. On the other hand, we know that $\mathsf{RootDensity}(C, \cdot)$ must eventually go to zero, by the ordinary Schwartz-Zippel lemma (Corollary 3.10). More precisely, the total degree of $C \in \mathbb{Z}[x_1, \ldots, x_n]$ is at most $2^{s(\lambda)}$, and hence for $r > 2^{\lambda \cdot s(\lambda)}$ we have $\mathsf{RootDensity}(C, r) < 2^{s(\lambda)}/2^{\lambda \cdot s(\lambda)} = 1/2^\lambda$, by Corollary 3.10.

In other words, we know that $\mathsf{RootDensity}(C, r)$ is nonnegligible for small r, and becomes negligible for large r. We now observe that $\mathsf{RootDensity}(C, r)$ cannot vary too fast with r. To wit, since the intervals $[r, 2r]$ and $[(r+1), 2(r+1)]$ differ (as sets) by at most a constant number of primes, the prime number theorem guarantees that for each $r > 0$, we have $|\mathsf{RootDensity}(C, r) - \mathsf{RootDensity}(C, r+1)| < \Theta(\lg r / r)$; and, for $r > 2^\lambda$, this difference is at most $\Theta(\lambda/2^\lambda)$. Since $\mathsf{RootDensity}(C, 2^\lambda) > \lambda^{-c}$ and $\mathsf{RootDensity}(C, 2^{\lambda \cdot s(\lambda)}) < 1/2^\lambda$, we conclude that for sufficiently large

λ, there exists some value r, with $2^\lambda < r < 2^{\lambda \cdot s(\lambda)}$, such that $\lambda^{-c}/3 < \mathsf{RootDensity}(C, r) < \lambda^{-c}/2$. For each $\lambda = \lambda_i$, let r_i be such a value r (chosen arbitrarily).

Finally, to reach a contradiction, we will show that for each $i \in \mathbb{N}$, when N is generated as a product of primes $p, q \leftarrow \mathsf{Primes}[r_i, 2r_i]$, the algorithm $\mathsf{CompSZFactor}(N)$ succeeds with probability $1/\mathrm{poly}(\lg r_i)$. First, since $N = pq \geq r_i^2$ and $r_i > 2^{\lambda_i}$, we have $\lambda_i < \lg N$, so the loop variable in $\mathsf{CompSZFactor}$ will take the value λ_i during some iteration. At this point, since $\lambda^{-c}/3 < \mathsf{RootDensity}(C, r) < \lambda^{-c}/2$, we know that with probability at least $\lambda^{-c}/3 \cdot (1 - \lambda^{-c}/2)$, the value z will be zero modulo p and not modulo q, and hence $\gcd(z, N)$ will be a proper factor of N. Since $\lambda < \lg r < \lambda \cdot s(\lambda)$, the success probability bound $\lambda^{-c}/3 \cdot (1 - \lambda^{-c}/2)$ is polynomial in $\lg r$, as desired. □

3.4 Proof Ingredient: Evaluation Constraints via Index Sets

Our main construction (Construction 3.1) uses index sets to enforce constraints on the adversary's evaluation. Notably, we aim to prevent the adversary from constructing zero-test queries that are inconsistent—i.e., use encodings that correspond to contradictory values of a given input bit x_i—as well as queries that correspond to superpolynomially many inputs $\mathbf{x}$, which would make our simulation procedure inefficient.

In this section, to show that our design of the index sets indeed prevents these undesired queries, we state and prove a few simple "structure lemmas", showing that all valid query polynomials have a certain form. To prove these structure lemmas, our techniques are similar to those of Brakerski and Rothblum [BR14] and Barak et al. [BGK+14], and we begin by adapting the definition of *input profiles* from the latter work. Intuitively, an input profile of a monomial t is a "bit mask" indicating, for each bit index $i \in [n]$, what value the monomial has chosen for the input bit x_i, based on its formal variables ($\hat{x}_{i,0}$ vs. $\hat{x}_{i,1}$, $\hat{u}_{i,0}$ vs. $\hat{u}_{i,1}$, and so on).

Definition 3.13 (Input Profiles ([BGK+14], adapted))**.** Fix a monomial t (with integer coefficient) over the formal variables of Construction 3.1 (Remark 2.11, Definition B.1). For each $i \in [n]$ and $b \in \{0, 1\}$, if t contains some formal variable in the list $(\hat{x}_{i,b}, \hat{u}_{i,b}, \hat{z}_{i,b}, \hat{w}_{i,b}, \hat{s}_{i,i',b,b'}, \hat{s}_{i',i,b',b})$ (for some $i' \in [n], b' \in \{0, 1\}$), then we say t *incorporates* b as its i^{th} bit.

If for some index $i \in [n]$, the monomial t incorporates both $b = 0$ and $b = 1$ as its i^{th} bit, then we define t's *input profile*, $\mathrm{prof}(t)$, to be $\perp$. Otherwise, we define $\mathrm{prof}(t)$ to be the string $b_1 \cdots b_n$, where, for each $i \in [n]$, b_i is the bit that t incorporates as its i^{th} bit, if any; and otherwise, $b_i = *$. We say a profile is *partial* if it contains the symbol $*$ at some position. We say that two input profiles r_1, r_2 *conflict* if for some i we have $\mathrm{prof}(r_1)_i \neq \mathrm{prof}(r_2)_i \in \{0, 1\}$. For non-conflicting input profiles $\mathrm{prof}(r_1), \mathrm{prof}(r_2)$ we say that their *merge* is the string whose i^{th} bit matches whichever of $\mathrm{prof}(r_1)_i, \mathrm{prof}(r_2)_i$ is not $*$, or if both are $*$, then $*$.

We extend this definition to polynomials r as follows. If for some monomial t in the formal expansion of r (without cancellation), we have $\mathrm{prof}(t) = \perp$, then we define $\mathrm{prof}(r) = \perp$. Otherwise, we define $\mathrm{prof}(r)$ to be $\{\mathrm{prof}(t) : t \text{ in the expansion of } r\}$.

Lemma 3.14 (Partial Terms Cannot Be Added)**.** *Fix monomials t_1, t_2 (with integer coefficients) over the formal variables of Construction 3.1 (Remark 2.11, Definition B.1). Suppose that* $\mathrm{prof}(t_1) \neq \mathrm{prof}(t_2)$*, neither* $\mathrm{prof}(t_1)$ *nor* $\mathrm{prof}(t_2)$ *is $\perp$, and at least one of the two is partial (Definition 3.13). Then t_1 and t_2 do not have the same index set.*

Proof. Without loss of generality, fix some $b \in \{0, 1\}$ such that $\mathrm{prof}(t_1)_i = b$ and $\mathrm{prof}(t_2)_i \neq b$. We note that t_2 cannot contain a factor of $\hat{x}_{i,b}$, $\hat{u}_{i,b}$, $\hat{z}_{i,b}$, $\hat{w}_{i,b}$, or $\hat{s}_{i,i',b,b'}$ (for $i' \in [n], b' \in \{0, 1\}$), since otherwise either $\mathrm{prof}(t_2)_i = b$ or $\mathrm{prof}(t_2) = \perp$. We consider the following cases.

- Suppose t_1 contains a factor of $\hat{z}_{i,b}$ or $\hat{w}_{i,b}$. Then its index set was formed using $\mathsf{BitCommit}_{i,b} = S_{i,b,i}$. Since t_2 contains neither $\hat{z}_{i,b}$ nor $\hat{w}_{i,b}$, its index set was not formed using $\mathsf{BitCommit}_{i,b} = S_{i,b,i}$. By construction of the straddling set system (Definition 2.20), we conclude that if t_1 and t_2 have the same index set, then this index set contains every element of $\mathcal{S}_i$. For each $i' \in [n]$, if $i' \neq i$, the only encodings that contain $S_{i,b,i'}$ are $(\hat{s}_{i,i',b,b'}, \hat{s}_{i',i,b',b})$ for some $b' \in \{0,1\}$. This implies that both t_1 and t_2 contain a factor of $\hat{s}_{i,i',b,b'}$ (for some $b' \in \{0,1\}$) for every $i' \in [n]$, contradicting our assumption that one of the two profiles is partial.

- Suppose t_1 does not contain a factor of $\hat{z}_{i,b}$ or $\hat{w}_{i,b}$, but does contain $\hat{s}_{i,i',b,b'}$ for some $i' \in [n], b' \in \{0,1\}$. Then its index set was formed using $S_{i,b,i'}$, while that of t_2 was not; and by the argument of the previous case, this contradicts our assumption that one of the two profiles is partial.

- Suppose t_1 does not contain a factor of $\hat{z}_{i,b}$, $\hat{w}_{i,b}$, or $\hat{s}_{i,i',b,b'}$. Then t_1 must contain one or more factors of $\hat{x}_{i,b}$ or $\hat{u}_{i,b}$. This implies that the degree of t_1's index set in the variable $X_{i,b}$ is nonzero. Since by assumption t_2 contains no factors of $\hat{x}_{i,b}, \hat{u}_{i,b}$, the only way for the index sets to match is if t_2 contains $\hat{C}^*$ or $\hat{z}_{i,1-b}$, and hence the common index set must contain Z_i. By assumption t_1 contains neither $\hat{z}_{i,b}$ nor $\hat{z}_{i,1-b}$, so it must contain $\hat{C}^*$. But this implies that the degree of $X_{i,b}$ in t_1's index set exceeds $\deg(x_i)$, so t_1's index set is not contained in $\mathcal{U}$, which is prohibited by definition. So we conclude that the two monomials t_1, t_2 have different index sets, as desired. □

Lemma 3.15 (Characterization of Input Profiles in Construction 3.1)**.** *Fix an efficient adversary $\mathcal{A}$ in the generic model of composite-order multilinear maps (Definition 2.10), and consider a formal polynomial r produced by $\mathcal{A}$, over the variables of Construction 3.1 (Remark 2.11, Definition B.2). The input profile* $\mathrm{prof}(r)$ *(Definition 3.13), is either (a) $\perp$, (b) a set of strings in $\{0,1\}^n$, none of which is partial; or (c) a singleton set whose one element is partial; and* $\mathrm{prof}(r)$ *can be computed efficiently by examining r. Further, the union of* $\mathrm{prof}(r)$ *over all formal polynomials r produced by $\mathcal{A}$ is a set of polynomial size.*

Proof. By induction on the sequence of formal polynomials r formed by $\mathcal{A}$ via oracle queries. For each query, forming a polynomial r, we consider the following cases.

- Suppose $r = r_1 + r_2$, for some formal polynomials r_1, r_2 already formed. If either $\mathrm{prof}(r_1)$ or $\mathrm{prof}(r_2)$ is $\perp$, then evidently $\mathrm{prof}(r) = \perp$, so we assume that neither $\mathrm{prof}(r_1)$ nor $\mathrm{prof}(r_2)$ is $\perp$. This leaves two cases.

  - Suppose that both $\mathrm{prof}(r_1)$ and $\mathrm{prof}(r_2)$ are sets of strings, none of which is partial. Then $\mathrm{prof}(r) = \mathrm{prof}(r_1) \cup \mathrm{prof}(r_2)$, and again none of the elements is partial.
  - Suppose that one of the two is a singleton set whose one element is partial (without loss of generality, suppose this is $\mathrm{prof}(r_1)$). Now, if $\mathrm{prof}(r_2) = \mathrm{prof}(r_1)$, then evidently $\mathrm{prof}(r) = \mathrm{prof}(r_2) = \mathrm{prof}(r_1)$. On the other hand, if $\mathrm{prof}(r_2) \neq \mathrm{prof}(r_1)$, then r_1 and r_2 contain two monomials in their formal expansions, resp., t_1 and t_2, such that $\mathrm{prof}(t_1) \neq \mathrm{prof}(t_2)$ and $\mathrm{prof}(t_1)$ is partial, contradicting Lemma 3.14.

- Suppose $r = r_1 r_2$, for some formal polynomials r_1, r_2 already formed. If either $\mathrm{prof}(r_1)$ or $\mathrm{prof}(r_2)$ is $\perp$, then evidently $\mathrm{prof}(r) = \perp$, so we assume that neither $\mathrm{prof}(r_1)$ nor $\mathrm{prof}(r_2)$ is $\perp$. This leaves two cases.

- Suppose that at least one of $\mathrm{prof}(r_1)$ and $\mathrm{prof}(r_2)$ is a set of polynomially many strings, none of which is partial (without loss of generality, suppose $\mathrm{prof}(r_1)$ satisfies this condition). Then we find that by definition, the profile of the result is either $\perp$ (if $\mathrm{prof}(r_2)$ is $\perp$ or contains any string that conflicts pairwise with one of $\mathrm{prof}(r_2)$), or else is just $\mathrm{prof}(r_1)$.
- Suppose $\mathrm{prof}(r_1)$ and $\mathrm{prof}(r_2)$ are each a singleton set whose one element is partial. Then we find that by definition, their product is either $\perp$ (if the two conflict), or else a singleton set (merging the two).

- Suppose $r = v$ for a formal variable v. Then $\mathrm{prof}(r)$ is a singleton set whose one element is $*$ at every position, except for the bit b at position i if v is one of the variables $(\hat{x}_{i,b}, \hat{z}_{i,b}, \hat{w}_{i,b})$.
- Suppose $r = c$ for a constant $c \in \mathbb{Z}$. Then $\mathrm{prof}(r)$ is a singleton set whose one element is $*$ at every position.

Finally, to establish the bound on the union over all polynomials produced, we note that the only case in which a new string appears in an input query is via the second case in a multiplication query, $r = r_1 r_2$, in which input profiles are merged. In this case, the merge occurs between two singleton sets, and so each query introduces at most one new string as an input profile. Since the adversary can make only polynomially many oracle queries the claim follows. □

Lemma 3.16 (Characterization of Zero-Test Polynomials in Construction 3.1)**.** *Fix an efficient adversary $\mathcal{A}$ in the generic model of composite-order multilinear maps (Definition 2.10), and consider a formal polynomial z produced by $\mathcal{A}$ at the top-level index set $\mathcal{U}$, over the variables of Construction 3.1 (Remark 2.11, Definition B.2). Any monomial t that occurs in the formal expansion of z (without cancellation) has one of the following two forms:*

1. *For some bits $x_1, \ldots, x_n \in \{0,1\}$, and constant $a \in \mathbb{Z}$, we have:*

$$t = a\hat{C}^* \left(\prod_{i \in [n]} \hat{w}_{i,x_i} \right) \left(\prod_{i_1 < i_2 \in [n]} \hat{s}_{i_1,i_2,x_{i_1},x_{i_2}} \right)$$

2. *For some bits $x_1, \ldots, x_n \in \{0,1\}$, and monomial function h, we have:*

$$t = h(\hat{x}_{1,x_1}, \ldots, \hat{x}_{n,x_n}, \hat{u}_{1,x_1}, \ldots, \hat{u}_{n,x_n}, (\hat{y}_j)_{j \in [m]}, \hat{v}) \left(\prod_{i \in [n]} \hat{z}_{i,x_i} \right) \left(\prod_{i_1 < i_2 \in [n]} \hat{s}_{i_1,i_2,x_{i_1},x_{i_2}} \right)$$

Proof. The claim follows by case analysis on the construction of the top-level index set $\mathcal{U}$ as the index set of the monomial t. Since $\mathcal{U}$ contains the index set $\prod_{i \in [n]} Z_i$, the monomial t must contain as factors some encodings yielding each Z_i, but the only such encodings are $\hat{C}^*$ and the $\hat{z}_{i,b}$. We consider two cases.

1. Suppose t contains $\hat{C}^*$ as a factor. Then the only remaining indices in $\mathcal{U}$ are the W_i and $\mathcal{S}_i$, which means the monomial t must contain only the variables $\hat{C}^*$ and some subset of the $\hat{w}_{i,b}, \hat{s}_{i,i',b,b'}$. Further, since each W_i appears exactly once in the top-level index set $\mathcal{U}$, the monomial t must contain exactly one of $(\hat{w}_{i,0}, \hat{w}_{i,1})$ for each $i \in [n]$. We define $x_i \in \{0,1\}$ so that t contains $\hat{w}_{i,x_i}$ for each $i \in [n]$. This implies that the index set of t contains $\mathsf{BitCommit}_{i,x_i} = S_{i,x_i,i}$. Since the index set of t is $\mathcal{U}$, which contains exactly one copy of

$\mathcal{S}_i \supset S_{i,x_i,i}$, the properties of straddling set systems (Corollary 2.21) imply that for each $i' \neq i \in [n]$, the monomial t contains exactly one factor whose index set contains $S_{i,x_i,i'}$. The only such variables are the $\hat{s}_{i,i',x_i,b'}, \hat{s}_{i',i,b',x_i}$ for some $b' \in \{0,1\}$; and by a symmetric argument, we conclude that for each $i < i' \in [n]$, the value of b' in the variable $\hat{s}_{i,i',x_i,b'}$ must be $x_{i'}$, establishing case (1) of the lemma.

2. Suppose t does not contain $\hat{C}^*$ as a factor. Then it must have obtained each index Z_i from some other encoding. The only such encodings are the $\hat{z}_{i,b}$, and since each Z_i appears exactly once in $\mathcal{U}$, the monomial t must contain exactly one of $(\hat{z}_{i,0}, \hat{z}_{i,1})$ for each $i \in [n]$. As in the previous case, define $x_i \in \{0,1\}$ so that t contains $\hat{z}_{i,x_i}$ for each $i \in [n]$. Now since the index set of $\hat{z}_{i,x_i}$ contains a factor of $X_{i,1-x_i}^{\deg(x_i)}$, and the top-level index set $\mathcal{U}$ contains only $\deg(x_i)$ copies of $X_{i,1-x_i}$, we conclude that for each $i \in [n]$, the monomial t does not contain any factors of $\hat{x}_{i,1-x_i}, \hat{u}_{i,1-x_i}$. In other words, it can be expressed as a monomial in the remaining encodings $(\hat{x}_{1,x_1}, \ldots, \hat{x}_{n,x_n}, \hat{u}_{1,x_1}, \ldots, \hat{u}_{n,x_n}, \hat{y}_j, \hat{v}, \hat{s}_{i,i',b,b'})$. Since the index set of t also contains $\mathsf{BitCommit}_{i,x_i} = S_{i,x_i,i}$ (via $\hat{z}_{i,x_i}$), by the argument of the previous case we conclude that the variables $\hat{s}_{i,i',b,b'}$ that occur in t are precisely $\hat{s}_{i_1,i_2,x_{i_1},x_{i_2}}$ for $i_1 < i_2$, satisfying case (2) of the lemma. □

Lemma 3.17. *Fix an efficient adversary $\mathcal{A}$. For every valid zero-test polynomial z produced by $\mathcal{A}$ in the security game for Construction 3.1 (in the generic model of composite-order multilinear maps), its input profile* $\mathrm{prof}(z)$ *is a set of strings in $\{0,1\}^n$, of polynomial size, none of which is partial. Further, this set can be computed efficiently by examining z.*

Proof. Immediate from Lemma 3.15 and Lemma 3.16. □

3.5 Main Proof

We are now equipped to prove the security theorems for our main construction (Theorems 3.5, 3.6, and 3.7). The three proofs are very similar, so we will present them all at once, with case analysis only when the approaches differ.

Proof of Theorems 3.5, 3.6, and 3.7.

Proof. Correctness follows by Lemma 3.2, while the efficiency of the obfuscator (as well as the succinctness property, for Theorem 3.5), follow by construction. To prove security, we now show how to construct an efficient simulator $\mathcal{S}$ (Definition 2.14).

Defining the simulator. Fix an efficient adversary $\mathcal{A}$. On input a universal circuit $C : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$ (along with oracle access to $C(\cdot, \mathbf{y})$), our simulator $\mathcal{S}$ will run internal copies of the obfuscator $\mathcal{O}$ (Construction 3.1) and the generic multilinear map oracle $\mathcal{M}$ (Definition 2.10). It will initialize $\mathcal{O}$ with a "dummy" secret input of all-zeroes, $\mathcal{O}^{\mathcal{M}}(1^\lambda, C, \mathbf{y} = 0^m)$, forward all of $\mathcal{O}$'s queries to its internal copy of $\mathcal{M}$, and forward $\mathcal{O}$'s output to $\mathcal{A}$ as the obfuscated circuit. Then, it will run $\mathcal{A}$, letting $\mathcal{M}$ answer all of $\mathcal{A}$'s oracle queries to $\mathsf{CM.Add}, \mathsf{CM.Mult}, \mathsf{CM.Encode}$. On a $\mathsf{CM.ZeroTest}$ query from $\mathcal{A}$ on a handle h, however, our simulator $\mathcal{S}$ will answer the query itself, as follows. It will examine the state of its internal copy of $\mathcal{M}$, and determine whether h refers to a formal polynomial z at some index set S (Remark 2.11, Definition B.2). If it does not, or if $S \neq \mathcal{U}$, then the simulator will immediately return $\perp$ as the answer to $\mathcal{A}$'s query. Otherwise, it will run the decision procedure of Figure 4, below, on the formal polynomial z, and answer $\mathcal{A}$ accordingly. Finally, when $\mathcal{A}$ terminates, the simulator will forward the output of $\mathcal{A}$ to the distinguisher D.

On input a formal polynomial z (represented efficiently as an arithmetic circuit):

1. Compute z's input profile, $\mathrm{prof}(z)$ (as specified in Definition 3.13). By Lemma 3.17, $\mathrm{prof}(z)$ is a polynomial-size set of strings in $\{0,1\}^n$, and it can be computed efficiently.
2. For each $\mathbf{x} = x_1 \cdots x_n \in \{0,1\}^n$ in $\mathrm{prof}(z)$, perform the following substitutions:
   (a) Let $z'_{\mathbf{x}} \in \mathbb{Z}[\hat{x}_{1,x_1}, \ldots, \hat{x}_{n,x_n}, \hat{y}_1, \ldots, \hat{y}_m]$ denote the original query z with its variables substituted as follows:

$$\hat{z}_{i,x_i}, \hat{w}_{i,x_i} \mapsto 1 \qquad \hat{z}_{i,1-x_i}, \hat{w}_{i,1-x_i} \mapsto 0$$

$$\hat{C}^* \mapsto C(\hat{x}_{1,x_1}, \ldots, \hat{x}_{n,x_n}, \hat{y}_1, \ldots, \hat{y}_m) \qquad \hat{u}_{i,b}, \hat{v}, \hat{s}_{i,b,i',b'} \mapsto 1$$

   Here, the term $C(\hat{x}_{i,x_1}, \ldots, \hat{x}_{i,x_n}, \hat{y}_1, \ldots, \hat{y}_m)$ denotes the arithmetic circuit for C, as an expression in terms of the formal variables $\hat{x}_{i,x_1}, \ldots, \hat{x}_{i,x_n}, \hat{y}_1, \ldots, \hat{y}_m$, "inlined" as a sub-circuit into the query.

   (b) Let $z''_{\mathbf{x}} \in \mathbb{Z}$ denote the original query z with its variables substituted as follows:[14]

$$\hat{C}^*, \hat{w}_{i,x_i} \mapsto 1 \qquad \hat{w}_{i,1-x_i} \mapsto 0 \qquad \hat{x}_{i,0}, \hat{x}_{i,1}, \hat{y}_i, \hat{z}_{i,0}, \hat{z}_{i,1} \mapsto 0 \qquad \hat{u}_{i,b}, \hat{v}, \hat{s}_{i,b,i',b'} \mapsto 1$$

   Now, perform the following tests.

   - If $z'_{\mathbf{x}} \not\equiv 0$, stop and answer "nonzero".
   - If $z'_{\mathbf{x}} \equiv 0$ and $z''_{\mathbf{x}} = 0$, go back to step 2(a) and continue with the next $\mathbf{x} \in \mathrm{prof}(x)$.
   - Otherwise, invoke the simulator's own oracle to determine whether $C(\mathbf{x}, \mathbf{y}) = 0$.
     - If $C(\mathbf{x}, \mathbf{y}) \neq 0$, stop and answer "nonzero".
     - If $C(\mathbf{x}, \mathbf{y}) = 0$, go back to step 2(a) and continue with the next $\mathbf{x} \in \mathrm{prof}(z)$.
3. Upon reaching the end of the iteration, answer "zero".

Figure 4: The simulator's decision procedure, to answer a CM.ZeroTest query on a handle referring to a formal polynomial z at the top-level index set $\mathcal{U}$.

Correctness of the simulator. By construction, for every secret key input $\mathbf{y}$, the obfuscation of $C(\cdot, \mathbf{y})$ consists of a list of encoded elements in the multilinear map, whose length depends only on the (public) universal circuit C. Since in the generic model these elements are represented by handles (independently uniform nonces), the adversary's initial input is independent of $\mathbf{y}$, and thus our simulator's dummy version (with $\mathbf{y} = 0^m$) is identical to the real distribution produced when $\mathcal{A}$ interacts directly with $\mathcal{M}$. The same argument applies to the responses to $\mathcal{A}$'s CM.Add,CM.Mult queries; while the response to a CM.Encode query will be $\perp$ in both the real game and in our simulated version (except for failure events with negligible probability), since the adversary cannot guess the nonce sp corresponding to the secret parameters of the multilinear map. Finally, we consider CM.ZeroTest queries on handles h. We call such a query *valid* if h refers to a formal polynomial z at the top-level index set $\mathcal{U}$ (Remark 2.11, Definition B.2), and we note that for queries that are not valid, the response will be $\perp$ by definition, in both our simulation and in the

real game.

Thus, it suffices to show that for each valid CM.ZeroTest query referring to a formal polynomial z, when the formal variables of z are instantiated with the values from this joint distribution from the real game (i.e., when z is answered by the actual multilinear map oracle), the answer matches that of the simulator with overwhelming probability. The result will then follow by a union bound, since the adversary can make only polynomially many queries.[15] We note that this analysis requires some care, since the moduli N_{ev} and N_{chk}, in addition to the encoded values, are hidden from the adversary. In particular, when we refer to the real distribution of a value in the oracle's table, we implicitly consider the joint distribution in the actual multilinear map oracle's table, over both the (hidden) primes composing N_{ev} and N_{chk} and the value itself (in $\mathbb{Z}_N^*$). (We assume without loss of generality that N_{ev} and N_{chk} are relatively prime and that all of the primes composing both moduli are distinct; by the birthday bound this holds with overwhelming probability over the coins of CM.Setup.)

Values in the real game. Fix a valid zero-test query polynomial z produced by the adversary during the game, and consider its formal expansion, after collecting like terms with respect to the variables $\hat{z}_{i,b}, \hat{w}_{i,b}, \hat{s}_{i_1,i_2,b_1,b_2}$. By Lemma 3.16, this expansion can be written as $z = \sum_{\mathbf{x} \in \text{prof}(z)} f_{\mathbf{x}}$, where each expression $f_{\mathbf{x}}$ has the following form:

$$f_{\mathbf{x}} \;=\; \left(h_{\mathbf{x}}(\hat{x}_{1,x_1}, \ldots, \hat{x}_{n,x_n}, \hat{u}_{1,x_1}, \ldots, \hat{u}_{n,x_n}, (\hat{y}_j)_{j \in [m]}, \hat{v}) \prod_{i \in [n]} \hat{z}_{i,x_i} \;+\; a_{\mathbf{x}} \hat{C}^* \prod_{i \in [n]} \hat{w}_{i,x_i} \right) \cdot \left(\prod_{i_1 < i_2 \in [n]} \hat{s}_{i_1,i_2,x_{i_1},x_{i_2}} \right)$$

for some constant $a_{\mathbf{x}} \in \mathbb{Z}$ and multivariate polynomial $h_{\mathbf{x}}$. We now consider the value taken by the polynomial $f_{\mathbf{x}}$ in the real game. To begin with, we note that the encodings $\hat{u}_{i,0}, \hat{u}_{i,1}, \hat{v}, \hat{s}_{i_1,i_2,b_1,b_2}$ are irrelevant, since they appear only as scaling factors in $f_{\mathbf{x}}$, and in the real value distribution they each take the value $1 \in \mathbb{Z}$. To formalize this fact, we define the following simplified expressions, $f'_{\mathbf{x}}$ and $h'_{\mathbf{x}}$, to be the result of substituting the formal constant 1 for the variables $\hat{u}_{i,0}, \hat{u}_{i,1}, \hat{v}, \hat{s}_{i,i',b,b'}$ in $f_{\mathbf{x}}$ (resp., $h_{\mathbf{x}}$), obtaining an expression of the following form:

$$f'_{\mathbf{x}} \;=\; h'_{\mathbf{x}}(\hat{x}_{1,x_1}, \ldots, \hat{x}_{n,x_n}, \hat{y}_1, \ldots, \hat{y}_m) \prod_{i \in [n]} \hat{z}_{i,x_i} \;+\; a_{\mathbf{x}} \hat{C}^* \prod_{i \in [n]} \hat{w}_{i,x_i}$$

Since the variables $(\hat{u}_{i,b}, \hat{v}, \hat{s}_{i,i',b,b'})$ all take the value 1 in both components ($\mathbb{Z}_{N_{\text{ev}}}$ and $\mathbb{Z}_{N_{\text{chk}}}$), the value of each $f'_{\mathbf{x}}$ is identically distributed to that of $f_{\mathbf{x}}$, and so it suffices to analyze the values of these simplified terms $f'_{\mathbf{x}}$.

Now, the first component (mod N_{ev}) of each $f_{\mathbf{x}} = f'_{\mathbf{x}}$ takes the following value in the real game:

$$f'_{\mathbf{x}} \;=\; h'_{\mathbf{x}}(x_1, \ldots, x_n, y_1, \ldots, y_n) \prod_{i \in [n]} \delta_{i,x_i} \mod N_{\text{ev}}$$

[14] We note that the integer $z''_{\mathbf{x}}$ is still represented as an arithmetic circuit, even though it has no remaining formal variables (after the substitutions). Thus, a simple way to test if $z''_{\mathbf{x}} = 0 \in \mathbb{Z}$ is to run the Schwartz-Zippel identity testing algorithm, i.e., evaluate the computation that produces $z''_{\mathbf{x}}$, modulo a random prime. This method works irrespective of the integer's bit length.

[15] Technically, we must also show that the distribution of the values in the oracle's table, conditioned on each possible sequence of past oracle query-response pairs (with no failure events), has negligible statistical distance from its prior distribution from CM.Setup; this follows by a standard conditional probability argument, given that the probability of each failure event is negligible.

while the second component (mod N_{chk}) takes the following value:

$$f'_{\mathbf{x}} \;=\; (h'_{\mathbf{x}} + a_{\mathbf{x}} C)\,(\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_n) \prod_{i \in [n]} \gamma_{i,x_i} \mod N_{\text{chk}}$$

Outcomes of the simulator's decision procedure. By construction, the simulator's response to each of the adversary's zero-test queries z (Figure 4) is a function only of the identity of z as a formal polynomial (except for failure events of negligible probability, i.e., failures of the ordinary Schwartz-Zippel algorithm during the decision procedure). We now walk through the simulator's substitutions for a valid zero-test query polynomial z, to show how the results depend on the real value distribution.

- The simulator's first substitution ($z'_{\mathbf{x}}$, Figure 4) is as follows:

  $$\hat{z}_{i,x_i}, \hat{w}_{i,x_i} \mapsto 1 \qquad \hat{z}_{i,1-x_i}, \hat{w}_{i,1-x_i} \mapsto 0$$

  $$\hat{C}^* \mapsto C(\hat{x}_{1,x_1}, \ldots, \hat{x}_{n,x_n}, \hat{y}_1, \ldots, \hat{y}_m) \qquad \hat{u}_{i,b}, \hat{v}, \hat{s}_{i,b,i',b'} \mapsto 1$$

  Intuitively, this substitution "zeroes out" most monomials in the expansion of z, preserving only those whose variables correspond to the input chosen $\mathbf{x} \in \text{prof}(z)$. The result of the substitution is the formal polynomial $z'_{\mathbf{x}} \equiv h'_{\mathbf{x}} + a_{\mathbf{x}} C$.

- The simulator's second substitution ($z''_{\mathbf{x}}$, Figure 4) is as follows:

  $$\hat{C}^*, \hat{w}_{i,x_i} \mapsto 1 \qquad \hat{w}_{i,1-x_i} \mapsto 0 \qquad \hat{x}_{i,0}, \hat{x}_{i,1}, \hat{y}_i, \hat{z}_{i,0}, \hat{z}_{i,1} \mapsto 0 \qquad \hat{u}_{i,b}, \hat{v}, \hat{s}_{i,b,i',b'} \mapsto 1$$

  Intuitively, this substitution preserves only the monomials whose variables correspond to the chosen input $\mathbf{x}$; and, of those, *only* the ones that correspond to the precomputed check values ($\hat{C}^* \prod_i \hat{w}_{i,x_i}$), not the ones from the adversary's own polynomial ($\hat{C} \prod_i \hat{x}_{i,x_i}$). The result of the substitution is an expression $z''_{\mathbf{x}}$ whose value is $a_{\mathbf{x}} \in \mathbb{Z}$.

Case analysis. Finally we consider the following cases, corresponding to the cases in the simulator's decision procedure (Figure 4), to show that the simulator's answer matches that of the real game with overwhelming probability.

- Suppose that for at least one $\mathbf{x} \in \text{prof}(z)$, we have $h'_{\mathbf{x}} + a_{\mathbf{x}} C \not\equiv 0$; and let $\mathbf{x}^*$ be the first such $\mathbf{x}$, in lexicographic order (so that $z'_{\mathbf{x}^*} \not\equiv 0$, and hence the simulator's response is "nonzero"). Then, modulo N_{chk}, the term $f'_{\mathbf{x}^*}$ takes the following value:

  $$f'_{\mathbf{x}^*} \;=\; (h'_{\mathbf{x}^*} + a_{\mathbf{x}^*} C)(\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_n) \prod_{i \in [n]} \gamma_{i,x^*_i} \mod N_{\text{chk}}$$

  We now claim that, with overwhelming probability, the value $(h'_{\mathbf{x}^*} + a_{\mathbf{x}^*} C)(\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_n)$ is invertible in $\mathbb{Z}_{N_{\text{chk}}}$. We consider two cases:

  - Suppose factoring is hard. We observe that $h'_{\mathbf{x}^*} + a_{\mathbf{x}^*} C$ is not identically zero as a function of its formal variables $\hat{x}_{1,x_1}, \ldots, \hat{x}_{n,x_n}, \hat{y}_1, \ldots, \hat{y}_m$, and yet can be computed by an arithmetic circuit of polynomial size (for example, the result of performing the simulator's substitution into the original query z). Thus, by the *computational* Schwartz-Zippel lemma (Lemma 3.12), along with a union bound over the primes composing N_{chk}, we conclude that with overwhelming probability its value is invertible modulo N_{chk}.

– Suppose factoring is not hard, but this is *not* the succinct variant of the construction (i.e., we are proving Theorem 3.6 or 3.7). In this case, we will bound the total degree of $h'_{\mathbf{x}^*} + a_{\mathbf{x}^*}C$. The total degree of C is at most 2^d, while the total degree of $h'_{\mathbf{x}^*}$ is at most that of $h_{\mathbf{x}^*}$, which in turn is at most $\sum_{i\in[n]} \deg(x_i) + \deg(\mathbf{y}) \leq 2^d$ (since its index set is a subset of $\mathcal{U}$). Thus $h'_{\mathbf{x}^*} + a_{\mathbf{x}^*}C$ has degree at most 2^d; it is not identically zero; and it can be computed by a poly-size arithmetic circuit (as in the previous subcase). By the Schwartz-Zippel lemma (Corollary 3.10), we conclude that the probability that it evaluates to zero on its uniformly random inputs over each prime-order field $\mathbb{Z}_p$ composing $\mathbb{Z}_{N_{\text{chk}}}$, with $p > 2^{d+\lambda}$, is at most $2^d n/2^{d+\lambda} = \text{negl}(\lambda)$, and again we conclude that with overwhelming probability its value is invertible modulo N_{chk}.

Finally, we consider the expansion of the entire query polynomial z, restricted to the second component (the ring $\mathbb{Z}_{N_{\text{chk}}}$), when all of its variables except the $\gamma_{i,b}$ are instantiated with their real value distribution. Since with overwhelming probability the expression $(h'_{\mathbf{x}^*} + aC)$ takes a value that is invertible modulo N_{chk}, the coefficient of the monomial $\prod_{i\in[n]} \gamma_{i,x_i^*}$ is nonzero in $\mathbb{Z}_p$ for each prime p composing N_{chk}, and hence the entire query polynomial z is not identically zero over this prime-order field (as a function of its remaining formal variables $\gamma_{i,b}$). Since its total degree in these variables is only $n = \text{poly}(\lambda)$, by the Schwartz-Zippel lemma we conclude that with overwhelming probability, the entire query z takes a value that is invertible in $\mathbb{Z}_{N_{\text{chk}}}$, and hence nonzero modulo $N = N_{\text{ev}}N_{\text{chk}}$ with overwhelming probability, Thus, with overwhelming probability, the simulator's response of "nonzero" is correct.

- Suppose that for every $\mathbf{x} \in \text{prof}(z)$, we have $h'_{\mathbf{x}} + a_{\mathbf{x}}C \equiv 0$, but for at least one $\mathbf{x} \in \text{prof}(z)$, we have $C(\mathbf{x}, \mathbf{y}) \neq 0$ and $a_{\mathbf{x}} \neq 0 \in \mathbb{Z}$. Let $\mathbf{x}^* = x_1^* \cdots x_n^* \in \{0,1\}^n$ be the first such $\mathbf{x}$, in lexicographic order (so that every $z'_{\mathbf{x}} \equiv 0$, but $z''_{\mathbf{x}^*} \neq 0$ and $C(\mathbf{x}^*, \mathbf{y}) \neq 0 \in \mathbb{Z}$, and hence the simulator's response is "nonzero"). Then since $h'_{\mathbf{x}^*} + a_{\mathbf{x}^*}C \equiv 0$, we have $h'_{\mathbf{x}^*}(\mathbf{x}^*, \mathbf{y}) = -a_{\mathbf{x}^*}C(\mathbf{x}^*, \mathbf{y}) \neq 0 \in \mathbb{Z}$. We now claim that with overwhelming probability, the value $h(\mathbf{x}^*, \mathbf{y})$ is invertible modulo N_{ev}. Intuitively, since $h(\mathbf{x}^*, \mathbf{y})$ is a nonzero integer, it must be invertible modulo N_{ev} (a product of random primes) with overwhelming probability. However, technically we must also establish that $h(\mathbf{x}^*, \mathbf{y})$ is not too large, and thus does not have too many prime factors. For simplicity, we just regard the value $h(\mathbf{x}^*, \mathbf{y}) \in \mathbb{Z}$ as a formal polynomial of degree zero, and follow the argument above. We consider the same two cases:

  – Suppose factoring is hard. Then $h'_{\mathbf{x}^*}(\mathbf{x}^*, \mathbf{y})$ is not identically zero, and yet can be computed by an arithmetic circuit of polynomial size (e.g., the result of the simulator's substitution). Thus, by the *computational* Schwartz-Zippel lemma (Lemma 3.12), along with a union bound over the primes composing N_{ev}, we conclude that with overwhelming probability the value $h'_{\mathbf{x}^*}(\mathbf{x}^*, \mathbf{y})$ is invertible modulo N_{ev}.
  – Suppose factoring is not hard, but this is *not* the succinct variant of the construction (i.e., we are proving Theorem 3.6 or 3.7). Then $h'_{\mathbf{x}^*}(\mathbf{x}^*, \mathbf{y})$ can be computed by an arithmetic circuit of degree at most 2^d (as above). By the Schwartz-Zippel lemma (Corollary 3.10), we conclude that the probability that its value is zero over each prime-order field $\mathbb{Z}_p$ composing $\mathbb{Z}_{N_{\text{ev}}}$, with $p > 2^{d+\lambda}$, is at most $1/2^{d+\lambda} = \text{negl}(\lambda)$, and again we conclude that with overwhelming probability the value $h'_{\mathbf{x}^*}(\mathbf{x}^*, \mathbf{y})$ is invertible modulo N_{ev}.

  The main result now follows as in the argument above (for the case $h'_{\mathbf{x}} + a_{\mathbf{x}}C \not\equiv 0$), with $\delta_{i,b}$ in place of $\gamma_{i,b}$, and N_{ev} in place of N_{chk}.

- Suppose that for every $\mathbf{x} \in \text{prof}(z)$, we have $h'_{\mathbf{x}} + a_{\mathbf{x}}C \equiv 0$ and either $a_{\mathbf{x}} = 0$ or $C(\mathbf{x}, \mathbf{y}) = 0$ (so that every $z'_{\mathbf{x}} \equiv 0$ and every $z''_{\mathbf{x}} = 0 \in \mathbb{Z}$, and hence the simulator's response is

"zero"). Then the value of $f'_{\mathbf{x}}$ modulo N_{chk} is zero, while the value of $f'_{\mathbf{x}}$ modulo N_{ev} is $h'_{\mathbf{x}}(x_1, \ldots, x_n, y_1, \ldots, y_n) \prod_{i \in [n]} \delta_{i,x_i}$. Now, since $h'_{\mathbf{x}} \equiv -a_{\mathbf{x}} C$, we have $h'_{\mathbf{x}}(\mathbf{x}, \mathbf{y}) = -a_{\mathbf{x}} C(\mathbf{x}, \mathbf{y})$. Since either $a_{\mathbf{x}} = 0$ or $C(\mathbf{x}, \mathbf{y}) = 0$, we also have $h'_{\mathbf{x}}(\mathbf{x}, \mathbf{y}) = 0$, and so the simulator's response of "zero" is correct with certainty.

Thus each of the simulator's responses is correct with overwhelming probability, and we conclude that the distribution of the adversary's output is statistically close to that in the real game, as desired. This concludes the main security proof. □

Remark 3.18 (Multi-Bit Output)**.** To avoid cluttering notation, we have phrased our main construction in terms of circuits C with single-bit output, i.e., $C : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$. However, we note that the extension to Boolean circuits multi-bit output is straightforward, and does not significantly increase the cost of any of the operations, since the values of intermediate wires can be reused. We now briefly outline this extension.

Consider a circuit with multi-bit output, $C : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^\ell$. In the single-bit case, Construction 3.1 (step 6) computes a single check value (and outputs its encoding, as part of the obfuscated program):

$$C^* = C(\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_m) \in \mathbb{Z}_{N_{\text{chk}}} \qquad \hat{C}^* = [0,\ C^*]_{Y^{\deg(\mathbf{y})} \prod_{i \in [n]} (X_{i,0} X_{i,1})^{\deg(x_i)} Z_i}$$

For the multi-bit case, we can modify the construction to compute ℓ such values (and output their encodings):

$$C^*_1 = C(\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_m)_1 \in \mathbb{Z}_{N_{\text{chk}}} \qquad \hat{C}^*_1 = [0,\ C^*_1]_{Y^{\deg(\mathbf{y})} \prod_{i \in [n]} (X_{i,0} X_{i,1})^{\deg(x_i)} Z_i}$$

$$\ldots$$

$$C^*_\ell = C(\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_m)_\ell \in \mathbb{Z}_{N_{\text{chk}}} \qquad \hat{C}^*_\ell = [0,\ C^*_\ell]_{Y^{\deg(\mathbf{y})} \prod_{i \in [n]} (X_{i,0} X_{i,1})^{\deg(x_i)} Z_i}$$

The proof of virtual black-box security proceeds as above, except we generalize the simulator's substitutions. In place of the $C^* \mapsto C(\hat{x}_{1,x_1}, \ldots, \hat{x}_{n,x_n}, \hat{y}_1, \ldots, \hat{y}_m)$, the simulator substitutes $C^*_k \mapsto C(\hat{x}_{1,x_1}, \ldots, \hat{x}_{n,x_n}, \hat{y}_1, \ldots, \hat{y}_m)_k$ for every $k \in [\ell]$; and in place of $C^* \mapsto 1$, the simulator substitutes $C^*_k \mapsto C(\mathbf{x}, \mathbf{y})_k \in \{0,1\}$ for every $k \in [\ell]$, using its own oracle to determine $C(\mathbf{x}, \mathbf{y})_k$ for each output bit k. In the index set lemmas (Section 3.4), the formal variables C^*_k now play the role of the previous single variable C^*, and we find that a zero-test query may have a monomial in each of the C^*_k (accompanied by factors of $\hat{w}_{i,x_i}$, as before). The analysis of the simulator now follows that of the main proof.

4 Performance Analysis and Applications

We now analyze the asymptotic efficiency of our main construction (Construction 3.1). Suppose we are obfuscating the keyed arithmetic circuit family $\mathcal{C} = (C_{\mathbf{y}})_{\mathbf{y} \in \{0,1\}^m}$, where each $C_{\mathbf{y}} = C(\cdot, \mathbf{y})$ and $C : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$ (the keyed "universal circuit") has size s and depth d. As in the construction, let $\deg(\mathbf{y})$ be the total degree of C in all of the variables $y_1, \ldots, y_m$; and for each $i \in [n]$ let $\deg(x_i)$ be the degree of C in the variable x_i. We also define $\Delta = \deg(\mathbf{y}) + \sum_{i \in [n]} \deg(x_i)$, and note that $\Delta < 2^d$.

In our analysis, for clarity, we measure the obfuscation size in terms of the number of ring elements, and the evaluation time in terms of the number of ring operations (i.e., encodings and operations in the multilinear map). In the CLT multilinear map [CLT13], which is currently

the only known instantiation for composite order, the element size grows with the square of the multilinearity degree required (though this bound could change with further development of the techniques). Thus, to obtain concrete performance measurements in terms of bits and Boolean operations using the CLT multilinear map, the reader should multiply every listed obfuscation size and evaluation time by the square of the degree of multilinearity. (For simplicity, we also drop factors of poly(λ) arising from the multilinear map implementation.)

4.1 Naive Analysis

Before we delve into the space of possible optimizations (Section 4.2), we present a simple analysis of our main construction as described above, to give a flavor of the relevant metrics.

Degree of multilinearity. The degree of any term (in the formal variables of the obfuscation) is bounded by the total degree of the top-level index set $\mathcal{U}$. In our case, this degree is $\Delta + 2n + n(2n-1) = O(\Delta + n^2) = O(2^d + n^2)$. (We note, however, that this bound does not incorporate the optimizations below, and under different optimizations the degree of multilinearity may increase; we discuss the details below.)

Obfuscation size. We now count the number of ring elements (in the multilinear map) that our construction requires. In an obfuscated circuit $\mathcal{O}(C_{\mathbf{y}})$, the encodings $\hat{x}_{i,b}, \hat{u}_{i,b}, \hat{z}_{i,b}, \hat{w}_{i,b}$ contribute a total of $8n$ elements; the encodings $\hat{y}_j, \hat{v}, \hat{C}^*$ contribute $m + 2$; and the final straddling-set fill encodings ($\hat{s}_{i_1,i_2,b_1,b_2}$) contribute an additional $4\binom{n}{2}$, for a total of $8n+m+2+4\binom{n}{2} = O(m+n^2)$ ring elements. (We remark that if we only need indistinguishability obfuscation, not virtual black-box, then we can omit the straddling-set fill encodings. In this case, the size of an obfuscation becomes $O(m+n)$ ring elements, and we can also reduce the additive $O(n^2)$ terms in the other parameters below; for details, we refer the reader to Appendix A.)

Evaluation time. To determine the number of ring operations required to evaluate the obfuscated program, the accounting is somewhat more involved. In particular, some operations will be required to "raise" intermediate elements via multiplication by unit encodings, when addition operations are required between elements with mismatching index sets. For clarity we first neglect these "raising" operations, and measure the total time required by all other phases of the algorithm. In this case, to evaluate the arithmetic circuit C on its encoded inputs, we require s ring operations in the multilinear map, one for each gate in C. We must then multiply the output, $\hat{C}$, by $\prod_{i\in[n]} \hat{z}_{i,x_i}$; multiply the testing term, $\hat{C}^*$, by $\prod_{i\in[n]} \hat{w}_{i,x_i}$; and subtract the results. This incurs $2n+1$ ring operations. We then multiply the entire result by $\prod_{i_1<i_2\in[n]} \hat{s}_{i_1,i_2,x_{i_1},x_{i_2}}$, incurring an additional $\binom{n}{2}$ ring operations, for a total of $s + 2n + 1 + \binom{n}{2} = O(s + n^2)$ ring operations (excluding the "raising" operations).

Finally, we account for the "raising" operations (for each of the s addition gates). We observe that in the naive evaluation algorithm, these operations completely dominate the time complexity. More precisely, for general circuits C, the two operands of an addition gate can each be encoded at an arbitrary subset of the index set $Y^{\deg(\mathbf{y})} \prod_{i\in[n]} X_{i,x_i}^{\deg(x_i)}$. If for each index X_{i,x_i}, Y we fill in the gap on both sides via repeated squaring, this results in at most $(\lg \deg(\mathbf{y}) + \sum_{i\in[n]} \lg \deg(x_i)) < (n+1)d$ ring operations per gate (plus the same quantity once more to precompute the table for repeated squaring). This would bring the total time complexity, using the naive algorithm, to $O(dns + n^2) = O(dns)$. To improve on this bound, we now describe a number of natural optimizations; Table 2 gives an overview of the corresponding efficiency improvements.

4.2 Optimizations

Optimization: cross-multiplication. During evaluation of the circuit (Figure 3), we can maintain two encodings for each wire, rather than just one. Specifically, if some intermediate wire takes the value a and is encoded at the index set S, then we maintain not only $[a]_S$, but also a separate unit encoding $[1]_S$, and propagate both in parallel. (We note that in the base case, for the circuit's inputs, we already have this additional unit encoding, by definition of our construction.)

Now, suppose we have two inputs to an addition gate: $([a]_S, [1]_S)$ and $([b]_T, [1]_T)$, for some index sets $S \neq T$. Rather than raising each index set to $S \cup T$, we can simply "cross-multiply" (raising the resulting index set to ST instead of $S \cup T$), by computing the two new encodings $[x]_S[1]_T + [1]_S[y]_T = [x+y]_{ST}$ and $[1]_S[1]_T = [1]_{ST}$. Multiplication is still straightforward, of course, since we can just compute $[x]_S[y]_T = [xy]_{ST}$ and $[1]_S[1]_T = [1]_{ST}$.

We now consider the effect of this optimization on the time complexity and the degree of multilinearity required. Perhaps surprisingly, addition gates are now more expensive than multiplication gates. Each addition gate now requires (at most) four ring operations (three multiplications and one addition). Thus, in total we require at most $4s + 2n + 1 + \binom{n}{2} = O(s + n^2)$ ring operations.

In order to implement this optimization, we find that the top-level index set must change. In particular, every gate (addition or multiplication) results in a *product* of intermediate sets (rather than a union), whether or not it increases the degree of the corresponding sub-circuit of C. Thus we can no longer bound the degree of multilinearity in terms of C's degree, as we did in the naive approach (Section 4.1). However, since the size of an index set at most doubles when cross-multiplication is performed on a gate of fanin 2, we can still conclude that the multilinearity degree is at most $O(2^d + n^2)$. In other words, our bound is the same in terms of the depth, but not in terms of the degree. Intuitively, this means that we pay the worst-case cost in multilinearity more often—under this optimization, every gate acts like a multiplication gate, whereas with the naive algorithm we could make use of the fact that not every circuit's degree is exponential in its depth.

Optimization: pre-mixing. In most cases, the "cross-multiplication" optimization above results in an excellent bound. However, under that optimization the multilinearity degree increases from $O(\Delta + n^2)$ to $O(2^d + n^2)$, which can be significant for circuits with many addition gates. In some cases—particularly in the setting of "noisy" multilinear maps, which is the only setting we know how to instantiate so far—minimizing the degree of multilinearity is paramount, even at the cost of increasing the number of ring operations required for evaluation. To this end, we now discuss another general-purpose optimization.

In this optimization, our goal is to eliminate the asymmetry in the mismatched index sets at each addition gate, so that each index set has equal degree in every X_i; this will enable us to run repeated squaring only twice for each gate (once on powers of all of the X's together, and once on the powers of Y), rather than $n+1$ times. To accomplish this, at the beginning of the evaluation, we "pre-mix" the encodings of each input variable $x_1, \ldots, x_n$. More specifically, for each $\hat{x}_{i,x_i}$, we form the encoding $\tilde{x}_{i,x_i} = \hat{x}_{i,x_i} \prod_{i' \neq i} \hat{u}_{i,x_i}$, and use $\tilde{x}_{i,x_i}$ in place of $\hat{x}_{i,x_i}$ during the circuit evaluation. By construction, each $\tilde{x}_{i,x_i}$ has the same value as $\hat{x}_{i,x_i}$, but now encoded at the index set $\prod_{i' \in [n]} X_{i',x'_i}$ (independent of i), rather than at the singleton index set X_{i,x_i}.

As for the time complexity, we can perform this "pre-mixing" in only $O(n)$ ring operations,[16] using a standard divide-and-conquer approach. First, assuming without loss of generality that $n = 2^r$ is a power of two, we define the *aligned intervals* to be those of the form $I_{k,\ell} = [k\ell+1, (k+1)\ell]$

[16] In fact, our algorithm for pre-mixing will not matter in the case of VBB obfuscation, since the overall cost will anyway be dominated by the $O(n^2)$ straddling-set unit encodings. However, in the case of $i\mathcal{O}$ (Appendix A), the difference between the $O(n^2)$ brute-force algorithm and the $O(n)$ divide-and-conquer algorithm becomes significant.

for some $\ell \in \{1, 2, 4, 8, \ldots, 2^{r-1}\}$ and $k \in [n/2^\ell]$; and their complements $\bar{I}_{k,\ell} = [n] \setminus I_{k,\ell}$. We define each interval's value to be the product of the corresponding unit encodings, $v_{k,\ell} = \prod_{i \in I_{k,\ell}} \hat{u}_{i,x_i}$ (and likewise $\bar{v}_{k,\ell}$ for the complements), and we observe that the desired outputs of the pre-mixing are just the complements' values, $\bar{v}_{0,1}, \ldots, \bar{v}_{n-1,1}$. Now, to perform the actual pre-mixing, we first precompute the value of each aligned interval, $v_{k,\ell}$, proceeding from the smallest length ℓ to the largest. Finally we compute the values of the complements, from the largest ℓ to the smallest, using the relations $\bar{v}_{2k,2^s} = \bar{v}_{k,2^{s+1}} \cdot v_{2k+1,2^s}$ and $\bar{v}_{2k+1,2^s} = \bar{v}_{k,2^{s+1}} \cdot v_{2k,2^s}$. Thus, the pre-mixing requires only $O(n)$ ring operations.

As an aside, we note that in the CLT multilinear map (Section 2.8), we can actually perform the "pre-mixing" computation in a much simpler way. Specifically, we can compute $\hat{U} = \prod_{i \in [n]} \hat{u}_{i,x_i}$, and just set $\tilde{x}_{i,x_i} = \hat{U} \cdot \hat{x}_{i,x_i} / \hat{u}_{i,x_i} \pmod{N_{\text{outer}}}$ for each $i \in [n]$—where N_{outer} is the CLT outer modulus, not the modulus N of the ring being represented in the multilinear map. However, this approach makes use of the concrete structure of ring elements in the CLT construction, and thus it cannot be performed in the generic model we adopt in this work. Indeed, if in the future we find a way to construct "clean" multilinear maps, the underlying representation may have a completely different algebraic structure, and these division operations may not be possible.

Regardless, the "pre-mixing" optimization we have just described enables us to run the repeated-squaring algorithm only twice for each gate (corresponding once to the powers of the index set $\prod_i X_{i,x_i}$ and once to those of Y), rather than $n+1$ times as in the naive approach. This brings the overall time complexity to $O(s \lg(n \sum_{i \in [n]} \deg(x_i)) + s \lg \deg(\mathbf{y})) + n^2) = O(s \lg(n\Delta) + n^2)$ ring operations. Like the "cross-multiplication" optimization above, this optimization also causes the multilinearity degree to increase (as compared with the naive algorithm), albeit less dramatically. Whereas the naive algorithm requires multilinearity degree $O(\Delta + n^2)$, here we require degree $O(n\Delta + n^2)$. (We could also bound the multilinearity degree in terms of d, as $O(2^d n + n^2)$, but in these terms, the cross-multiplication optimization above is strictly better—we emphasize that this "pre-mixing" optimization is designed for circuits with very low degree, $\Delta \ll 2^d$.)

	Degree of multilinearity	Obfuscation size (# ring elements)	Evaluation time (# ring operations)
Naive algorithm	$O(\Delta + n^2)$	$O(m + n^2)$	$O(dns)$
Optimization: pre-mixing	$O(n\Delta + n^2)$	$O(m + n^2)$	$O(s \lg(n\Delta) + n^2)$
Optimization: cross-mult.	$O(2^d + n^2)$	$O(m + n^2)$	$O(s + n^2)$

Table 2: Performance of our main construction, both with the naive algorithm (Section 4.1) and with various optimizations (Section 4.2), for a keyed arithmetic circuit $C : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$ of size s, depth d. Here $\Delta = \deg(\mathbf{y}) + \sum_{i \in [n]} \deg(x_i)$, where $\deg(\mathbf{y})$ is C's total $\mathbf{y}$-degree, and $\deg(x_i)$ is its degree in x_i. We note that $n, m, \Delta \leq 2^d$, and $n, m < s$. We present the cost here in terms of ring elements and ring operations. The concrete cost in bits and bit operations depends on the performance of the multilinear map (Section 2.8); for "clean" maps (whose existence is still open), the cost is just $\text{poly}(\lambda)$, while for the CLT scheme [CLT13], the reader should multiply every obfuscation size and evaluation time by $O(\deg^2) \cdot \text{poly}(\lambda)$, where deg is the corresponding multilinearity degree from the first column.

Circuit-specific optimizations. The optimizations we describe above are *general-purpose*: they can be applied to any circuit family, regardless of its structure. If we take into account the structure of specific circuits C, this kind of optimization process can take us much further. For example, in the "pre-mixing" optimization above, we still require $O(\lg(n\Delta))$ ring operations for every addition gate in the circuit, since the index sets of the two inputs may differ by any subset of the index set $Y^{\deg(\mathbf{y})} \prod_i X_{i,x_i}^{\deg(x_i)}$. Intuitively, this reflects the fact that the circuit may have many "stale" access instructions, in which a wire feeding into some gate originated much higher in the circuit.

Many circuits of interest do not have this unfortunate property. For instance, suppose the circuit C represents the execution trace of a parallel RAM program which maintains a memory of w items, and performs d sequential steps.[17] Further, suppose that each step consists entirely of either pairwise multiplications, pairwise additons, or "mux" operations (Section 2.2) with secret key input bits; and without loss of generality, suppose every item is updated at every step (inserting dummy gates and items as needed).

For such a circuit, using the "pre-mixing" optimization above, we do not need to raise *any* intermediate index sets, because all elements of the memory have the same index set at every step. In this case, rather than $O(\lg(n\Delta))$, the number of ring operations per gate becomes constant. (We also remark that this model is well-suited to many cryptographic operations; for example, block ciphers such as AES are defined as a sequence of simple transformations on a state that is maintained throughout the execution.)

The observation we have just described is only one example of a circuit-specific optimization. In a broader sense, we see a rich space of potential optimizations, which are particularly effective for *keyed* circuit families in which the structure of the circuit C is known. In most applications, we believe it does not make sense to use general-purpose obfuscation, throwing away any knowledge of the computation's structure—just as it usually does not make sense to make a machine's computation oblivious by brute-force iteration over its entire memory (Section 2.2).

4.3 Examples

We now specialize our obfuscation construction to several standard computational models, in order to provide a direct comparison with other approaches.

4.3.1 Example: Keyed Circuits

We begin with the natural setting of general, keyed arithmetic circuits (straight-line programs, Section 2.3) $C : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$ of size s and depth d, with input length n and key length m. (We remark that the analysis here also covers the case of Boolean circuits, since it is straightforward to substitute arithmetic gates for Boolean gates.) Using the "cross-multiplication" optimization (Section 4.2), we obtain a multilinearity degree of $O(2^d + n^2)$, obfuscation size $O(m + n^2)$ ring elements, and evaluation time $O(s + n^2)$ ring operations (Table 3).

In our construction, the number of ring elements and ring operations required is polynomial, while in all other known approaches, it is exponential in d. This new bound reemphasizes the importance of the open question of "clean" multilinear maps (Section 2.7). Currently, with "noisy" maps, the size of a ring element itself grows with the degree of multilinearity. If we could construct clean maps, then the element size—and hence the cost of the entire obfuscation—would be independent of the degree, and thus polynomial in the size of the original circuit, even for P/poly.

[17] We also require that its execution is oblivious in the strong sense of Section 2.2.

	Degree of multilinearity	Obfuscation size (# ring elements)	Evaluation time (# ring operations)
Via Barrington's Thm. [GGH+13b, BR14, BGK+14]	$O(4^d n + n^2)$	$O(4^d n + n^2)$	$O(4^d n + n^2)$
[AGIS14]	$O(2^d + n^2)$	$O(8^d + n^2)$	$O(8^d + n^2)$
[AGIS14] + [Gie01]	$O(2^{(1+\varepsilon)d} + n^2)$	$O(2^{(1+\varepsilon)d} 4^{2/\varepsilon} + n^2)$	$O(2^{(1+\varepsilon)d} 4^{2/\varepsilon} + n^2)$
This work	$O(2^d + n^2)$	$O(m + n^2)$	$O(s + n^2)$

Table 3: Performance for (keyed) circuits of input length n, key length m, size s, and depth d. For most applications of interest, we have $n, s \ll 2^d$; and we always have $n, s < O(2^d)$, since the gates have fanin two. For this table we refer to the version of our construction with the "cross-multiplication" optimization (Section 4.2). We present the cost here in terms of ring elements and ring operations. The concrete cost in bits and bit operations depends on the performance of the multilinear map (Section 2.8); for "clean" maps (whose existence is still open), the cost is just $\text{poly}(\lambda)$, while for the CLT scheme [CLT13], the reader should multiply every obfuscation size and evaluation time by $O(\deg^2) \cdot \text{poly}(\lambda)$, where deg is the corresponding multilinearity degree from the first column.

Motivation: obfuscation for PRFs. One of the most compelling applications of obfuscation—and, in particular, for the setting of keyed circuits—is to obfuscate a PRF. By the work of Garg et al. [GGH+13b], obfuscation for a log-depth PRF, along with fully homomorphic encryption (FHE), suffices to bootstrap NC^1 obfuscation to obfuscation for P/poly. From a more practical perspective, a VBB-obfuscated PRF is a noninteractive analog of a trusted third party capable of performing cryptographic operations, which makes it a very useful primitive. Indeed, one of the most striking applications is that of *short signatures*—namely, λ-bit signatures with $\approx\lambda$-bit security—by the following construction (adapted from a result of Sahai and Waters [SW14]):

$$\mathsf{pk} \;=\; \mathcal{O}[\,(m,\sigma) \mapsto \text{PRF}(\mathsf{sk}, m) \stackrel{?}{=} \sigma\,]$$

We do not prove formally that this construction yields an (adaptively) secure signature scheme, since technically even the virtual black-box property is not enough to argue this directly; we would need an interactive analog of VBB in the $\mathcal{M}$-idealized model, similar to the signature security game itself. We leave formalizing such definitions for future work; here, we only consider the construction as a motivating example, assuming a notion of obfuscation that is "strong enough".

Dream example: obfuscating AES. Suppose that our goal is to construct an obfuscated PRF that is implementable in practice. With previous approaches, based on branching programs, this would immediately rule out the choice of the AES block cipher as the PRF, since its depth is on the order of hundreds (including, among other features, 10 rounds of the AES S-box), and thus the branching program size, $\sim 4^d n$, would be astronomical. With our new approach, however (Table 3), the number of ring elements and operations is polynomial in the size of the original circuit, regardless of its depth (as shown in Section 4.3.1). Thus, for the first time, the only remaining obstacle to *implementable* obfuscation for AES (and other moderately-deep circuits of interest) is the noise growth in known multilinear maps.

Suppose for the moment that we had "clean" multilinear maps, so that the noise growth, and hence the degree of multilinearity, were irrelevant. In this case, the analysis for obfuscated AES would proceed as follows. For simplicity, we will actually obfuscate the keyed Boolean function $V_{\text{AES}}((\mathbf{x}, \boldsymbol{\sigma}), K)$ which is 1 if AES-128$(K, \mathbf{x}) = \boldsymbol{\sigma}$, and 0 otherwise (as might be desired in other

cryptographic applications).[18] Using the "cross-multiplication" optimization of Section 4.2, the parameters are as follows.

Obfuscating the function $V_{\text{AES}}((\mathbf{x}, \boldsymbol{\sigma}), K) := (\text{AES-128}(K, \mathbf{x}) \stackrel{?}{=} \boldsymbol{\sigma})$:

- Input length: $n = 256$ bits ($\mathbf{x}, \boldsymbol{\sigma} \in \{0,1\}^{128}$).
- Key length: $m = 128$ bits ($K \in \{0,1\}^{128}$).
- Obfuscation size: 132738 ring elements (or 2178 for $i\mathcal{O}$; see Appendix A).
  - We require $8n + m + 2 + 4\binom{n}{2} = 132738$ ring elements. We note for the weaker notion of indistinguishability obfuscation, we can omit the $4\binom{n}{2}$ straddling-set unit encodings; see Appendix A. In this case, we only require $8n + m + 2 = 2178$ ring elements.
- Evaluation time: 280255 ring operations (or 247615 for $i\mathcal{O}$; see Appendix A).
  - The work of Kreuter, shelat, and Shen [KSS12] shows that AES-128 can be implemented with a Boolean circuit consisting of 30728 gates. To this circuit, we append a "check" subcircuit that tests whether the output of AES equals the second input $\boldsymbol{\sigma} \in \{0,1\}^{128}$. This can be done with 128 XNOR gates, followed by a tree of 127 AND gates.
  - Using the "cross-multiplication" optimization (Section 4.2), an AND gate can be implemented using 2 ring operations, and no Boolean gate requires more than 8 ring operations.[19] Thus, the main body of the evaluation requires at most $8 \cdot 30728 = 245824$ ring operations, while the "check" subcircuit at the end requires $8 \cdot 128 + 2 \cdot 127 = 1278$ ring operations. Finally, computation of the resulting element $\hat{C} \prod_{i \in [n]} \hat{z}_{i,x_i} - \hat{C}^* \prod_{i \in [n]} \hat{w}_{i,x_i}$ requires an additional $2n + 1 = 513$ ring operations, while multiplication by the correct straddling-set unit encodings $\hat{s}_{i_1,i_2,x_{i_1},x_{i_2}}$ requires $\binom{n}{2} = 32640$ ring operations, for a total of $245824 + 1278 + 513 + 32640 = 280255$ ring operations.

Of course, even though the obfuscation size and evaluation time here are relatively tractable (in terms of ring elements and ring operations), the degree of multilinearity is still proportional to 2^{d+8} where d is the depth of AES—i.e., still astronomical. We reemphasize that even with our new techniques, it remains infeasible to obfuscate AES, but now *only* because we do not know how to construct clean multilinear maps. The results of this section provide further motivation for this important open question.

4.3.2 Example: NC^1 Circuits (Unkeyed, Balanced Boolean Formulas)

We now treat the class of (unkeyed) NC^1 circuits: i.e., Boolean circuit families $C : \{0,1\}^n \to \{0,1\}$ of size n^c and depth $d = k \lg n$ for constants $c, k > 0$, over the basis $\{\text{AND}, \text{OR}, \text{NOT}\}$ (with fanin

[18] We remark that while we could also obfuscate AES itself, we would run into technical difficulties due to the fact that its output is 128 bits, rather than a single bit. While it would be straightforward to modify our construction to permit this (Remark 3.18), we use a single-bit example here to avoid cluttering the analysis.

[19] To see this, let $f : \{0,1\} \times \{0,1\} \to \{0,1\}$ be a Boolean function, and consider $|f^{-1}(1)|$, i.e., the number of inputs that map to 1 under f. If $|f^{-1}(1)| \in \{0,4\}$, then f is a constant function and requires no ring operations. If $|f^{-1}(1)| \in \{1,3\}$, then f requires up to two negations (of the inputs), followed by an AND, perhaps followed by another negation, for a total of at most 5 ring operations. Finally, if $|f^{-1}(1)| = 2$, then either f is independent of one of the input bits (and hence requires no ring operations), or else either $f \in \{\text{XOR}, \text{XNOR}\}$. If $f = \text{XOR}$, then $f(a,b) = a + b - 2ab$, and we require 4 ring operations to produce the new encodings of $1, a, b, ab$, followed by 3 operations to produce the sum, for a total of 7. If $f = \text{XNOR}$, then $f(a,b) = (1-a)$ XOR b, for a total of 8.

2 and unbounded fanout). This class coincides with the class of polynomial-size *balanced* Boolean formulas, i.e., formulas whose abstract syntax forms a balanced binary tree. This is also the setting of other constructions [GGH+13b, BR14, BGK+14] that construct obfuscation via Barrington's theorem [Bar86].

To adapt our construction to this setting, given an unkeyed family of log-depth circuits, we will need to convert it to a keyed family. To do this, we first duplicate gates in the Boolean circuit as necessary to make the fanout 1 (as in the transformation to a balanced formula); we note that the number of inputs of the resulting circuit is at most 2^d, and thus its size is at most $\sum_{\ell \in [d]} 2^\ell = O(2^d)$. By DeMorgan's laws, we re-express the circuit in terms of only AND and OR gates in its inputs and their negations (this requires $O(2^d)$ additional steps at the beginning of evaluation to compute the negation of each variable). Now, to convert each Boolean gate (a AND b or a OR b) to a *keyed* arithmetic gate, we replace it by the arithmetic expression $(y_1 - a)(y_2 - b)y_3 + y_4$, where y_1, y_2, y_3, y_4 are new secret key input bits in $\mathbf{y}$. For an AND gate, these inputs are $(0, 0, 1, 0)$, while for an OR gate, they are $(1, 1, -1, 1)$. This results in $4 \cdot 2^{d+1} = O(2^d)$ secret key input bits.

Further, since we must also hide the identity of each input to the circuit among $x_1, \ldots, x_n$, we preface the main circuit with a "pre-muxing", similar to our "pre-mixing" step from above. More specifically, for each $k \in [2^d]$, we introduce secret key input bits $y_{k,1}, \ldots, y_{k,n}$, of which all are 0 except one which is 1 (determining which of the n input variables $x_1, \ldots, x_n$ should actually feed into the k^{th} input to the Boolean circuit). In other words, the k^{th} Boolean input is now written as the arithmetic circuit $\sum_{i \in [n]} y_{k,i} x_i$, as in the "mux" operation of Section 2.2, and we have introduced an additional $O(2^d n)$ key inputs. We also require $O(2^d n)$ ring operations here to compute these sums, including raising operations by unit encodings to make the index sets match (as described in the "pre-mixing" optimization, Section 4.2).

We also note that the raising operations required to compute the "mux" sums, $\sum_{i \in [n]} y_{k,i} x_i$, have already accomplished the effect of the "pre-mixing" optimization—namely, every resulting input to the Boolean circuit has the same index set, $Y \prod_{i \in [n]} X_{i,x_i}$. Since the main body of the Boolean circuit is balanced, it will continue to be the case that within each layer of the circuit, every encoding has the same index set, and that index set has equal degree in every X_{i,x_i}. Now, the only remaining addition operations are in the affine functions $(y_1 - a)(y_2 - b)y_3 + y_4$ for wires a, b, for which there is an index set discrepancy between y_1, y_2, y_4, at index set Y; and $a, b, (y_1 - a)(y_2 - b)y_3$, each at index set $Y^r (\prod_{i \in [n]} X_{i,x_i})^s$ for some r, s. Since these values r, s are purely a function of the depth of the current layer, we can resolve the discrepancy by maintaining only a constant number of unit encodings in parallel with the circuit's evaluation (as in the "cross-multiplication" optimization, Section 4.2) to resolve the discrepancy. Thus, the time complexity is dominated by the $O(2^d n)$ operations in the initial pre-muxing.

The degree of the resulting arithmetic circuit in each key input in $\mathbf{y}$ is 1 (since each is only multiplied into the main circuit once, and our main computation has fanout 1), and thus its total $\mathbf{y}$-degree is $\deg(\mathbf{y}) = O(2^d n)$. Its degree in each input in $\mathbf{x}$ is precisely 2^d, and thus its total $\mathbf{x}$-degree is $O(2^d n)$. Overall, the resulting scheme requires multilinearity degree $O(2^d n + n^2)$, the obfuscation size is $O(2^d n + n^2)$ ring elements, and the evaluation time is $O(2^d n + n^2)$ ring operations.[20]

4.3.3 Example: Unbalanced Boolean Formulas

In the previous section, we considered the case of *balanced* Boolean formulas, which correspond directly to NC^1 circuits. Most Boolean formulas of interest are not balanced, however, but rather

[20] We remark that we could optimize the obfuscation size further, reducing the $O(2^d n)$ to $O(2^d \lg n)$ by representing each input's identity in binary during the pre-muxing step, but we describe the unary case here to keep the presentation simple.

	Degree of multilinearity	Obfuscation size (# ring elements)	Evaluation time (# ring operations)
Via Barrington's Thm. [GGH+13b, BR14, BGK+14]	$O(4^d n + n^2)$	$O(4^d n + n^2)$	$O(4^d n + n^2)$
[AGIS14]	$O(2^d n + n^2)$	$O(8^d n + n^2)$	$O(8^d n + n^2)$
[AGIS14] + [Gie01]	$O(2^{(1+\varepsilon)d} n + n^2)$	$O(2^{(1+\varepsilon)d} 4^{2/\varepsilon} n + n^2)$	$O(2^{(1+\varepsilon)d} 4^{2/\varepsilon} n + n^2)$
This work	$O(2^d n + n^2)$	$O(2^d n + n^2)$	$O(2^d n + n^2)$

Figure 5: Performance for balanced Boolean formulas (unkeyed NC^1 circuits, Section 4.3.2), of depth d and hence size $O(2^d)$, with $n \leq 2^d$ input variables. We present the cost here in terms of ring elements and ring operations. The concrete cost in bits and bit operations depends on the performance of the multilinear map (Section 2.8); for "clean" maps (whose existence is still open), the cost is just $\mathrm{poly}(\lambda)$, while for the CLT scheme [CLT13], the reader should multiply every obfuscation size and evaluation time by $O(\deg^2) \cdot \mathrm{poly}(\lambda)$, where deg is the corresponding multilinearity degree from the first column.

have depth comparable to their size. In this section, we study the performance of our construction in the setting of arbitrary Boolean formulas. (By convention, we call these formulas *unbalanced*, although we do not exclude formulas that happen to be balanced.)

While it is known how to balance Boolean formulas [PM76], transforming an arbitrary Boolean formula of size s to a balanced formula of depth $O(\lg s)$ (at which point the results above suffice), Ananth et al. [AGIS14] show that there are much more efficient ways to obfuscate arbitrary Boolean formulas. However, even the constructions of [AGIS14] still pay the overhead of converting these formulas to matrix branching programs, in order to randomize those programs via Kilian's protocol. In this section, we describe a more direct approach, making use of the properties of our new construction.

Keyed, unbalanced formulas. First, we consider the case of *keyed* formulas, in which the formula itself is public, but some of its variables are secret key inputs. Fix a keyed Boolean formula of size s. We evaluate the formula directly (regarding it as a circuit with fanout 1, and converting Boolean gates to the corresponding arithmetic gates), and we use the "cross-multiplication" optimization (Section 4.2) to implement the raising operations. With this solution, the degree of each subformula's index set (counting multiplicity) remains proportional to the size of the subformula, and thus the required multilinearity degree is $O(s + n^2)$. By the analysis above, the obfuscation size is also $O(s + n^2)$ ring elements, and evaluation requires $O(s + n^2)$ ring operations.

Unkeyed, unbalanced formulas: solution 1. The case of general (unkeyed) Boolean formulas is particularly interesting, because in order to make the computation *keyed* (and thus suitable for our techniques), we need to design an oblivious algorithm (Section 2.2) for evaluating these formulas. To do this, we proceed as follows.

Suppose we have an input Boolean formula ϕ of size s. First, we consider the formula in postfix order, so that, e.g., the formula $\phi = (x \wedge y) \vee z$ becomes the string "$x\, y \wedge z \vee$". Second, we describe a standard stack-based evaluation algorithm: maintain a stack of size s, and execute a series of s steps, one for each token in the (postfix) formula. Upon encountering a variable, push its value onto the stack; upon encountering an operator, pop the top two elements and push the operator's result.

In this algorithm, it is easy to make everything oblivious except the action of the stack. Specif-

	Degree of multilinearity	Obfuscation size (# ring elements)	Evaluation time (# ring operations)
[AGIS14]	$O(s + n^2)$	$O(s^3)$	$O(s^3)$
[AGIS14] + [Gie01]	$O(s^{1+\varepsilon} + n^2)$	$O(s^{1+\varepsilon}4^{2/\varepsilon} + n^2)$	$O(s^{1+\varepsilon}4^{2/\varepsilon} + n^2)$
This work	$O(s + n^2)$	$O(s + n^2)$	$O(s + n^2)$

Table 4: Performance for keyed, unbalanced Boolean formulas (Section 4.3.3) of size s, with n input variables (where $n < s$). We present the cost here in terms of ring elements and ring operations. The concrete cost in bits and bit operations depends on the performance of the multilinear map (Section 2.8); for "clean" maps (whose existence is still open), the cost is just $\text{poly}(\lambda)$, while for the CLT scheme [CLT13], the reader should multiply every obfuscation size and evaluation time by $O(\deg^2) \cdot \text{poly}(\lambda)$, where deg is the corresponding multilinearity degree from the first column.

ically, for each of s steps, our algorithm performs each possible action in parallel (i.e., fetch the value of each input variable; evaluate OR; evaluate AND). Then, we introduce new secret key input bits indicating which operation should actually be performed, according to the formula being obfuscated, and "mux" the results (Section 2.2). As for the action of the stack, evidently we could make it oblivious by brute force: maintain an array of size $O(s)$, and at each step, perform the push and pop operations in parallel at each of the $O(s)$ possible positions of the top of the stack; introduce new secret key input bits, corresponding to the actual position of the top of the stack at each step; and "mux" the results accordingly (Section 2.2). This would require $O(s)$ operations at each of the $O(s)$ steps, bringing the time complexity of all of the stack operations to $O(s^2)$.

To improve on this solution, we will replace the brute-force implementation of an oblivious stack with a recursive construction, which yields an oblivious stack of capacity s whose amortized complexity is only $O(\lg s)$ physical operations per logical operation (push or pop). Such a construction follows from the classic Turing machine simulation paradigm of Hennie and Stearns [HS66] and Pippenger and Fischer [PF79], as adapted by Mitchell and Zimmerman to the setting of general-purpose data structures [MZ14]. In more detail, the oblivious stack construction of [MZ14] consists of a series of "mux" operations on the blocks of memory that store the stack items. To implement a total of t logical operations (push or pop), the number of block-wise "mux" operations required is $O(t)$, and the number of individual bit muxes (totalling over all block-wise operations) is $O(t \lg s)$. Using this efficient oblivious stack implementation, we find that the s required push/pop operations require only $O(s \lg s)$ steps. Further, since the algorithm is oblivious (Section 2.2), it can be translated directly to a (keyed) straight-line program over $\mathbb{Z}$.

We now come to the question of how to implement the "raising" operations (Sections 4.1, 4.2), which will decide the overall optimization strategy. Since every operation except the AND/OR gates is a mux (which has degree 1 in each operand, and degree 2 overall), we might expect that the overall degree would be polynomial, and therefore we should use the "pre-mixing" optimization (Section 4.2), to keep the multilinearity degree low. Unfortunately, the AND/OR gates pose a problem, since they require multiplications not between the algorithm's state and the secret key input bits, but rather among values within the algorithm's state itself (as in repeated squaring). This means that regardless of which optimization we use ("pre-mixing" or "cross-multiplication"), the multilinearity degree becomes exponential in s. This would not a problem for "clean" multilinear maps, but it limits the solution's applicability for the "noisy" maps that are known at present. For simplicity, however, we first analyze this solution as-is, tolerating the degree blowup (as would make sense for "clean" maps). Below we will consider alternative approaches to reduce the multilinearity degree.

For our analysis here, however, we use the simple "cross-multiplication" optimization (Section 4.2). The computation depth d is is $O(s)$, since the oblivious stack of [MZ14] requires only $O(1)$ (multi-bit) muxes, amortized, for each of the $O(s)$ steps of the algorithm. So the required degree of multilinearity is $2^{O(s)}$—making this solution suited mainly to "clean" multilinear maps, as discussed above. To bound m, the number of secret key input bits, we note that for each of the $O(s)$ evaluation steps we require $O(1)$ key bits, amortized, to decide the outcomes of the $O(1)$ muxes (some of which are multi-bit); plus $O(\lg n)$ for the initial "pre-muxing" (Section 4.3.2) to load the value of the correct input in $x_1, \ldots, x_n$, representing the input's identity in binary. This is a total of $m = O(s \lg n)$ input bits, and so the obfuscation size is $O(m + n^2) = O(s \lg n + n^2)$ ring elements. As for the evaluation time, for each of the $O(s)$ steps we require $O(n)$ operations to pre-mux the input variables, while the operations of the oblivious stack require $O(\lg s)$ operations (amortized) per step, for a total evaluation time of $O(ns + s \lg s)$ ring operations.

Unkeyed, unbalanced formulas: solution 2. The exponential degree of the previous solution is not coincidental. Since our evaluation algorithm is oblivious, it does not know the arithmetic degree of each subformula at the time of evaluation. It only knows that, for each of s steps, it is drawing two values from the stack, performing a pairwise multiplication (to evaluate an AND/OR gate), and potentially writing the result back into the stack. In other words, as far as the algorithm knows, it could be multiplying the same operands over and over, effectively performing repeated squaring, and thereby computing a function of degree 2^s. Of course, we know the input formula cannot really act this way: for a formula of size s, the actual degree of every subformula is at most s. What is reflected in the index sets, however, is this conservative approximation (or "static analysis") of the program's operation, which results in the required multilinearity degree of $2^{O(s)}$.

Since we know the actual degree of every term is at most s, we can consider the following alternative approach based on dynamic programming. In the previous solution, throughout the execution, every value in memory had equal degree in each input x_i (but, of course, this degree increased exponentially with the time step). Now, in this modified solution, we replace every value v in memory with an s-tuple of values, $(v_1, \ldots, v_s)$, where for each $k \in [s]$, the index set of the value v_k has degree k in every X_{i,x_i}. During the execution, each s-tuple $(v_1, \ldots, v_s)$ may store the value v taken by some subformula of degree $\delta < s$; in this case, we specify that v_δ holds the *actual* value of v (0 or 1), and every other $v_{\delta'}$ holds the value 0 (for $\delta' \neq \delta \in [s]$). With this modification, each AND/OR operation on values popped from the stack is no longer a scalar operation, but rather becomes a *convolution* between two s-tuples of ring elements. For example, to execute the operation $z \leftarrow v$ AND w on the corresponding s-tuples $(z_1, \ldots, z_s), (v_1, \ldots, v_s), (w_1, \ldots, w_s)$, we would need to set $z_i = \sum_{j \in [i-1]} v_j w_{i-j}$, to capture all possible degrees $(j, i-j)$ of the subformulas corresponding to the values v, w.

To analyze this solution, first suppose that we perform these convolutions by brute force, in $O(s^2)$ multiplications each. We use the "pre-mixing" optimization (Section 4.2), and we also precompute tables of unit encodings at index sets $(\prod_i X_{i,x_i})^1, \ldots, (\prod_i X_{i,x_i})^{\deg(x_i)}$ and $Y^1, \ldots, Y^{\deg(\mathbf{y})}$, so that each addition then requires only $O(1)$ ring operations to raise the operands' index sets. The resulting evaluation time is $O(ns)$ ring operations to pre-mux the n inputs for the $O(s)$ steps (Section 4.3.2);[21] plus $O(s^3)$ for the $O(s)$ brute-force convolutions; plus $O(s^2 \lg s)$ for the $O(s)$ stack operations (now on s-tuples instead of scalars); plus the final $O(n^2)$, as always, for the straddling-set fill elements. Since $n < s$, the cost of the brute-force convolution dominates, and the overall evalua-

[21] Even though we now represent the input bits' identities in $[n]$ in binary, we can still perform the pre-muxing in $O(n)$ ring operations, via a standard divide-and-conquer approach like the one described in Section 4.2 for the "pre-mixing" optimization.

tion time is $O(s^3)$ ring operations. The obfuscation size is $O(m+n^2) = O(s \lg n + s + n^2) = \tilde{O}(s+n^2)$ ring elements (since we represent the identity of each input in $[n]$ in binary, requiring a total of $s \lg n$ secret key input bits; plus $O(s)$ additional bits to decide each of the multi-bit muxes of the oblivious stack). Finally, to bound the multilinearity degree, we argue that at the end of time step t, taking the maximum over all s-tuples in memory, for each $k \in [s]$ the index set of the k^{th} component of any tuple has degree $O(k(t+n))$. Indeed, during each time step t, muxing with the input bits never increases the resulting degree above $O(n)$; the $O(1)$ multi-bit muxes for the oblivious stack only contribute $O(1)$ factors of Y to each index set; and the convolution preserves the desired dependence on the index $k \in [s]$. Thus, the overall multilinearity degree is $O(s(s+n)) = O(s^2)$.

Unkeyed, unbalanced formulas: solution 3. At this point, it seems natural to improve the evaluation time of solution 2, by replacing the brute-force $O(s^2)$ convolution algorithm with an $O(s \lg s)$ solution based on the Fast Fourier Transform (FFT). This turns out to be problematic. For example, in the Fourier transform of $(v_1, \ldots, v_s)$, the first component is the value $v_1 + \ldots + v_s$—where each v_k is supplied to us via an element whose index set has degree k in every index X_{i,x_i}. It is not clear how to obtain the sum, except by first raising all of the elements $v_1, \ldots, v_s$ to a common index set, which would have degree at least s (in every X_{i,x_i}). Yet, from the Fourier transform we would like to extract the elements of the convolution, somehow obtaining for each $k \in [s]$ the k^{th} result at an index set whose degree (in every X_{i,x_i}) is only $k < s$.

However, if we are willing to work outside the generic model, using the properties of the concrete CLT instantiation, we can make this idea work. Since the CLT ring elements are represented concretely as integers, with the ring operations corresponding to addition and multiplication modulo N_{outer} (Section 2.8), we can use any efficient FFT-based algorithm on the representations themselves, over the integers, and reduce each result modulo N_{outer} at the end. For each of the $O(s)$ steps of the formula evaluation, this amounts to a fast multiplication of two $O(s \lg(s{N_{\text{outer}}}^2))$-bit integers, followed by s reductions modulo N_{outer}. Since $\lg N_{\text{outer}}$ grows with the multilinearity degree in the CLT scheme, we have $\lg N_{\text{outer}} = \Omega(s)$, and so the cost of the modular reductions dominates that of the FFT. Thus the total asymptotic cost for the convolutions, summing over all s steps of the formula evaluation, is at most the cost of $O(s^2)$ ring operations, reducing the overall evaluation time to that of $O(ns + s^2 \lg s + s^2 + n^2) = O(s^2 \lg s)$ ring operations.

5 Conclusions and Open Problems

We have proposed a new way to obfuscate programs, using composite-order multilinear maps. Our construction operates directly on straight-line programs (arithmetic circuits), rather than converting them to matrix branching programs, and thereby achieves considerable improvements in efficiency, as well as exposing a rich new design space of oblivious algorithms to serve as input to the obfuscator. Our results also yield the first known obfuscator (for keyed circuit families) in which the number of ring elements depends only on the lengths of the input and of the secret key.

Our results in this work highlight a number of open problems for further study. For one, our construction relies on the fact that the multilinear map has (hidden) composite order, in order to implement encodings of direct products via the Chinese Remainder Theorem. It is natural to wonder whether this property can be emulated using standard prime-order multilinear maps, via composite-to-prime-order transformations. While such transformations are known in some settings [GLW14, HHH+14], we are not aware of any transformations for *asymmetric* multilinear maps, in which we use index sets from arbitrary subset lattices with multiplicity (Section 2.5). We leave this as an interesting open problem for future work.

	Degree of multilinearity	Obfuscation size (# ring elements)	Evaluation time (# ring operations)
[AGIS14]	$O(ns)$	$O(ns^3)$	$O(ns^3)$
[AGIS14] + [Gie01]	$O(ns^{1+\varepsilon})$	$O(ns^{1+\varepsilon}4^{2/\varepsilon})$	$O(ns^{1+\varepsilon}4^{2/\varepsilon})$
This work (sol.1)	$2^{O(s)}$	$\tilde{O}(s+n^2)$	$\tilde{O}(ns)$
This work (sol.2)	$O(s^2)$	$\tilde{O}(s+n^2)$	$O(s^3)$
This work (sol.3) (concrete + [CLT13])	$O(s^2)$	$\tilde{O}(s+n^2)$	$\tilde{O}(s^2)$

Table 5: Performance for unkeyed, unbalanced Boolean formulas (Section 4.3.3), of size s, with n input variables (where $n < s$). Our first two solutions work for any instantiation of multilinear maps, while our last solution achieves better performance by making use of the concrete representations in the CLT construction. We present the cost here in terms of ring elements and ring operations. The concrete cost in bits and bit operations depends on the performance of the multilinear map (Section 2.8); for "clean" maps (whose existence is still open), the cost is just $\text{poly}(\lambda)$, while for the CLT scheme [CLT13], the reader should multiply every obfuscation size and evaluation time by $O(\deg^2) \cdot \text{poly}(\lambda)$, where deg is the corresponding multilinearity degree from the first column.

Another compelling line of research concerns the security assumptions and the applicability of the generic model. As Brakerski and Rothblum observe [BR14], no multilinear map can possibly instantiate the generic model perfectly, since we are able to use the generic model to construct VBB obfuscation, which we know is impossible for general circuit families [BGI+01]. Moreover, our results in this work highlight the fact that there are simple concrete examples of differences between the generic model and its instantiation via the CLT scheme—for instance, in Solution 3 of Section 4.3.3, based on the Fast Fourier Transform, our computation is valid for CLT encodings but cannot be implemented in the generic model. While this particular difference is fortuitous, we are led to consider whether there are other algebraic properties that hold in the CLT scheme—and may, in fact, be compatible with concrete security assumptions, such as that of [GLW14]—yet which may indicate fundamental weaknesses in the generic model as it is used here and in [GGH+13b, BR14, BGK+14]. We do not know of any such weaknesses at present, but further cryptanalysis is needed. On the positive side, it would also be useful to avoid relying on the generic model entirely, instead proving $i\mathcal{O}$ for our construction based on concrete, instance-independent assumptions [GLW14, GLSW14]. We leave this as another important problem for future work.

The central open problem: "clean" multilinear maps. This work eliminates a key obstacle to implementing obfuscation in practice. Since we no longer depend on converting circuits to branching programs, our construction would extend immediately to obfuscation for P/poly, with reasonable parameters—except for the noise growth in known, "noisy" multilinear maps. Our results demonstrate that the question of "clean" multilinear maps is not a technicality, but a fundamental open problem.

6 Acknowledgements

The author is grateful to Dan Boneh, Amit Sahai, and David J. Wu for many helpful comments and discussions. This work was supported by an NSF Graduate Research Fellowship, the DARPA PROCEED program, a grant from ONR, and an IARPA project provided via DoI/NBC. Opinions,

findings and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of DARPA or IARPA.

References

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

[AGIS14] Prabhanjan Ananth, Divya Gupta, Yuval Ishai, and Amit Sahai. Optimizing obfuscation: Avoiding Barrington's theorem. 2014. `http://eprint.iacr.org/`.

[AM09] Divesh Aggarwal and Ueli M. Maurer. Breaking RSA generically is equivalent to factoring. In *EUROCRYPT*, 2009.

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

[BCP14] Elette Boyle, Kai-Min Chung, and Rafael Pass. On extractability obfuscation. In *TCC*, 2014.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In *CRYPTO*, 2001.

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

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

[BGN05] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-dnf formulas on ciphertexts. In *TCC*, 2005.

[BL97] D. Boneh and R. J. Lipton. Effect of operators on straight line complexity. In *ISTCS*, 1997.

[BR14] Zvika Brakerski and Guy N. Rothblum. Virtual black-box obfuscation for all circuits via generic graded encoding. In *TCC*, 2014.

[BS03] Dan Boneh and Alice Silverberg. Applications of multilinear forms to cryptography. *Contemporary Mathematics*, 324(1), 2003.

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: definitions and challenges. In *TCC*, 2011.

[CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited (preliminary version). In *STOC*, 1998.

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

[CV13] Ran Canetti and Vinod Vaikuntanathan. Obfuscating branching programs using black-box pseudo-free groups. Cryptology ePrint Archive, Report 2013/500, 2013. `http://eprint.iacr.org/`.

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

[DL78] Richard A. DeMillo and Richard J. Lipton. A probabilistic remark on algebraic program testing. *Inf. Process. Lett.*, 7(4), 1978.

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

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

[GGH+13b] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, 2013.

[GGH14] Craig Gentry, Sergey Gorbunov, and Shai Halevi. Graded multilinear maps from lattices. Cryptology ePrint Archive, Report 2014/645, 2014. `http://eprint.iacr.org/`.

[GGHZ14] Sanjam Garg, Craig Gentry, Shai Halevi, and Mark Zhandry. Fully secure attribute based encryption from multilinear maps. Cryptology ePrint Archive, Report 2014/622, 2014. `http://eprint.iacr.org/`.

[Gie01] Oliver Giel. Branching program size is almost linear in formula size. *J. Comput. Syst. Sci.*, 63(2), 2001.

[GLSW14] Craig Gentry, Allison B. Lewko, Amit Sahai, and Brent Waters. Indistinguishability obfuscation from the multilinear subgroup elimination assumption. Cryptology ePrint Archive, Report 2014/309, 2014. `http://eprint.iacr.org/`.

[GLW14] Craig Gentry, Allison B. Lewko, and Brent Waters. Witness encryption from instance independent assumptions. In *CRYPTO*, 2014.

[GR07] Shafi Goldwasser and Guy N. Rothblum. On best-possible obfuscation. In *TCC*, 2007.

[HHH+14] Gottfried Herold, Julia Hesse, Dennis Hofheinz, Carla Ràfols, and Andy Rupp. Polynomial spaces: A new framework for composite-to-prime-order transformations. In *CRYPTO*, 2014.

[HS66] F. C. Hennie and Richard Edwin Stearns. Two-tape simulation of multitape Turing machines. *J. ACM*, 13(4), 1966.

[Jou00] Antoine Joux. A one round protocol for tripartite Diffie-Hellman. In *ANTS*, ANTS-IV, London, UK, 2000. Springer-Verlag.

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

[KSS12] Benjamin Kreuter, Abhi Shelat, and Chih-Hao Shen. Billion-gate secure computation with malicious adversaries. In *USENIX Security Symposium*, 2012.

[Mil04] Victor S Miller. The Weil pairing, and its efficient calculation. *Journal of Cryptology*, 17(4), 2004.

[MOV93] Alfred Menezes, Tatsuaki Okamoto, and Scott A. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. *IEEE Transactions on Information Theory*, 39(5), 1993.

[MZ14] John C. Mitchell and Joe Zimmerman. Data-oblivious data structures. In *STACS*, 2014.

[O'N10] Adam O'Neill. Definitional issues in functional encryption. Cryptology ePrint Archive, Report 2010/556, 2010. `http://eprint.iacr.org/`.

[PF79] Nicholas Pippenger and Michael J. Fischer. Relations among complexity measures. *J. ACM*, 26(2), 1979.

[PM76] Franco P. Preparata and David E. Muller. Efficient parallel evaluation of boolean expression. *IEEE Trans. Computers*, 25(5), 1976.

[PST14] Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation from semantically-secure multilinear encodings. In *CRYPTO*, 2014.

[Sch80] J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4), October 1980.

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In *EUROCRYPT*, 1997.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *STOC*, 2014.

[Zip79] Richard Zippel. Probabilistic algorithms for sparse polynomials. In *EUROSAM*, 1979.

A Indistinguishability Obfuscation

In addition to the original definition of virtual black-box obfuscation [BGI$^+$01], Barak et al. introduce a weaker notion called *indistinguishability obfuscation* ($i\mathcal{O}$), for which the (standard-model) negative results do not apply. In this appendix, we show that our main construction (Construction 3.1) can be modified to achieve better parameters, if we only need $i\mathcal{O}$ rather than VBB (in the generic model of multilinear maps).

We now present the formal definition of $i\mathcal{O}$, due to Barak et al. [BGI$^+$01]. Intuitively, $i\mathcal{O}$ says that an obfuscated version of a circuit C leaks no more information than any circuit, of the same size as C, that computes the same function. (For a more precise formulation of this analogy, we refer the reader to the work of Goldwasser and Rothblum [GR07].)

Definition A.1 (Indistinguishability Obfuscation in an $\mathcal{M}$-Idealized Model ([BGI$^+$01], adapted))**.** Let $\mathcal{C} = (\mathcal{C}_\lambda)_{\lambda\in\mathbb{N}}$ be a family of Boolean circuits, and let $\mathcal{M}$ be a stateful oracle (possibly randomized). We say that a PPT machine $\mathcal{O}$ is an *indistinguishability obfuscator* for $\mathcal{C}$ in the $\mathcal{M}$-idealized model, if the following conditions are satisfied.

- Correctness: There is a negligible function ε such that for all $\lambda \in \mathbb{N}$, every circuit $C \in \mathcal{C}_\lambda$, every input $\mathbf{x}$ to C, and all possible random coins for $\mathcal{M}$, we have

$$\Pr[(\mathcal{O}^{\mathcal{M}}(1^\lambda, C))(\mathbf{x}) \neq C(\mathbf{x})] \ < \ \varepsilon(\lambda) \ ,$$

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

- Indistinguishability: For every PPT adversary $\mathcal{A}$, there is a negligible function ε such that for every $C_0, C_1 \in \mathcal{C}_\lambda$ such that C_0 and C_1 compute the same function and $|C_0| = |C_1|$, we have

$$\left|\Pr[\mathcal{A}^{\mathcal{M}}(\mathcal{O}^{\mathcal{M}}(1^\lambda, C_0))) = 1] - \Pr[\mathcal{A}^{\mathcal{M}}(\mathcal{O}^{\mathcal{M}}(1^\lambda, C_1))) = 1]\right| \ < \ \varepsilon(\lambda) \ ,$$

  where the probability is over the coins of $\mathcal{A}, \mathcal{O}$, and $\mathcal{M}$.

We also specialize Definition A.1 to the classes P/poly and NC^1, just as in Definitions 2.15 and 2.16. Since the details are identical, we omit the formal definitions.

We now show that if we omit the "straddling-set" elements from our main construction (Construction 3.1), we can obtain significantly better parameters, while still satisfying the weaker notion of $i\mathcal{O}$. For completeness, we now state the modified construction.

Construction A.2 (Indistinguishability Obfuscation for Keyed Circuits)**.** Let $\mathsf{CM} = (\mathsf{CM.Setup}, \mathsf{CM.Add}, \mathsf{CM.Mult}, \mathsf{CM.ZeroTest}, \mathsf{CM.Encode})$ be a composite-order multilinear map (Definition 2.8). Fix an input $(C, \mathbf{y})$, where $\mathbf{y} \in \{0,1\}^m$, and $C : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$ is an arithmetic circuit (representing the keyed circuit $C_{\mathbf{y}}$, as in Section 2.4). (We note that we can obtain keyed circuit families from various other machine models, including general Boolean circuits, by the transformations of Section 2.4.)

As in Construction 3.1, let d be the depth of the circuit C; let $\deg(\mathbf{y})$ be the total degree of C in all of the variables $y_1, \ldots, y_m$; and for each $i \in [n]$ let $\deg(x_i)$ be the degree of C in the variable x_i. For a security parameter $\lambda \in \mathbb{N}$ (represented in unary), the obfuscation procedure $\mathcal{O}(1^\lambda, C, \mathbf{y})$ operates as follows.

$\underline{\mathcal{O}(1^\lambda, C, \mathbf{y})}$:

1. Construct the following index set of fresh formal symbols (Definition 2.7) as the top-level index set:

$$\mathcal{U} = Y^{\deg(\mathbf{y})} \prod_{i \in [n]} (X_{i,0} X_{i,1})^{\deg(x_i)} Z_i W_i$$

2. Run $(\mathsf{CM.pp}, \mathsf{CM.sp}, N_{\mathrm{ev}}, N_{\mathrm{chk}}) \leftarrow \mathsf{CM.Setup}(\mathcal{U}, 1^\lambda, 2)$.

3. For each $i \in [n]$, generate uniformly random values $\alpha_i, \gamma_{i,0}, \gamma_{i,1} \leftarrow \mathbb{Z}^*_{N_{\mathrm{chk}}}$ and $\delta_{i,0}, \delta_{i,1} \leftarrow \mathbb{Z}^*_{N_{\mathrm{ev}}}$. For each $j \in [m]$, generate a uniformly random value $\beta_j \leftarrow \mathbb{Z}^*_{N_{\mathrm{chk}}}$.

4. Compute the value $C^* = C(\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_m) \in \mathbb{Z}_{N_{\mathrm{chk}}}$.

5. Using $\mathsf{CM.Encode}(\mathsf{CM.sp}, \cdot)$, for $i \in [n]$, $j \in [m]$, and $b \in \{0,1\}$, generate the following encoded ring elements (using the notation of Remark 2.9):

$$\hat{x}_{i,b} = [b, \alpha_i]_{X_{i,b}} \qquad \hat{u}_{i,b} = [1,1]_{X_{i,b}} \qquad \hat{y}_j = [y_j, \beta_j]_Y \qquad \hat{v} = [1,1]_Y$$

$$\hat{z}_{i,b} = [\delta_{i,b}, \gamma_{i,b}]_{X_{i,1-b}^{\deg(x_i)} Z_i W_i} \qquad \hat{w}_{i,b} = [0, \gamma_{i,b}]_{W_i}$$

$$\hat{C}^* = [0,\ C^*]_{Y^{\deg(\mathbf{y})} \prod_{i\in[n]} (X_{i,0}X_{i,1})^{\deg(x_i)} Z_i}$$

We refer to the elements $\hat{u}_{i,b}, \hat{v}$ as *unit encodings*, since they each encode $1 \in \mathbb{Z}_N$, and they are incorporated solely for their effect on the index sets.

6. Output the values above, along with the public parameters of the multilinear map:

$$\mathcal{O}(1^\lambda, C, \mathbf{y}) \quad = \quad \left(\ \mathsf{CM.pp}, \ \ (\hat{x}_{i,b}, \hat{u}_{i,b}, \hat{z}_{i,b}, \hat{w}_{i,b})_{i\in[n],b\in\{0,1\}}, \ \ (\hat{y}_j)_{j\in[m]}, \ \ \hat{v}, \ \ \hat{C}^* \ \right)$$

To evaluate the obfuscated program $\tilde{C}_{\mathbf{y}} = \mathcal{O}(1^\lambda, C, \mathbf{y})$ on an input $\mathbf{x} = x_1 \cdots x_n \in \{0,1\}^n$, the evaluation procedure $\mathcal{O}.\mathrm{Eval}(\tilde{C}_{\mathbf{y}}, C, \mathbf{x})$ operates exactly as in Construction 3.1, except we omit the final unit encoding $\hat{\sigma}$ (as well as the computation to produce it).

To simplify the presentation, for $i\mathcal{O}$ we only state the result for succinct obfuscation from "clean" multilinear maps. To adapt the construction to "noisy" maps, or to eliminate the assumption that factoring is hard, we would modify the proof in the same way as in the case of VBB (Section 3.5).

Theorem A.3. *Suppose that factoring is hard (Assumption 3.11). Then Construction A.2 achieves succinct indistinguishability obfuscation for P/poly in the generic model of clean composite-order multilinear maps.*

To prove Theorem A.3, we will still need "structure lemmas" as in Section 3.4, in which we show that the index sets of terms in the construction force all of the adversary's valid zero-test queries to take a certain form. For $i\mathcal{O}$, our proofs of these lemmas will be much simpler, since we need not deal with the "straddling-set" commitment encodings $\hat{s}_{i_1,i_2,b_1,b_2}$; but the results of the lemmas will be correspondingly weaker, and in particular we can no longer conclude that each query's input profile, $\mathrm{prof}(z) \subset \{0,1\}^n$, contains only polynomially many inputs. As we will see, if our goal is only to prove $i\mathcal{O}$ and not VBB, then these weaker lemmas suffice.

Lemma A.4 (Characterization of Zero-Test Polynomials in Construction A.2)**.** *Fix an efficient adversary $\mathcal{A}$ in the generic model of composite-order multilinear maps (Definition 2.10), and consider a formal polynomial z produced by $\mathcal{A}$ at the top-level index set $\mathcal{U}$, over the variables of Construction A.2 (Remark 2.11, Definition B.2). Any monomial t that occurs in the formal expansion of z (without cancellation) has one of the following two forms:*

1. *For some bits $x_1, \ldots, x_n \in \{0,1\}$, and constant $a \in \mathbb{Z}$, we have:*

$$t = a\hat{C}^* \left(\prod_{i\in[n]} \hat{w}_{i,x_i} \right)$$

2. *For some bits $x_1, \ldots, x_n \in \{0,1\}$, and monomial function h, we have:*

$$t = h(\hat{x}_{1,x_1}, \ldots, \hat{x}_{n,x_n}, \hat{u}_{1,x_1}, \ldots, \hat{u}_{n,x_n}, (\hat{y}_j)_{j\in[m]}, \hat{v}) \left(\prod_{i\in[n]} \hat{z}_{i,x_i} \right)$$

Proof. The proof follows that of Lemma 3.16, without the encodings $\hat{x}_{i,i',b,b'}$. We proceed by case analysis on the construction of the top-level index set $\mathcal{U}$ as the index set of the monomial t. Since $\mathcal{U}$ contains $\prod_{i\in[n]} Z_i$, the monomial t must contain as factors some encodings yielding each Z_i, but the only such encodings are $\hat{C}^*$ and the $\hat{z}_{i,b}$. We consider two cases.

1. Suppose t contains $\hat{C}^*$ as a factor. Then the only remaining indices in $\mathcal{U}$ are the W_i. which means the monomial t must contain only the variables $\hat{C}^*$ and some subset of the $\hat{w}_{i,b}$. Further, since each W_i appears exactly once in the top-level index set $\mathcal{U}$, the monomial t must contain exactly one of $(\hat{w}_{i,0}, \hat{w}_{i,1})$ for each $i \in [n]$. We define $x_i \in \{0,1\}$ so that t contains $\hat{w}_{i,x_i}$ for each $i \in [n]$, establishing case (1) of the lemma.

2. Suppose t does not contain $\hat{C}^*$ as a factor. Then it must have obtained each index Z_i from some other encoding. The only such encodings are the $\hat{z}_{i,b}$, and since each Z_i appears exactly once in $\mathcal{U}$, the monomial t must contain exactly one of $(\hat{z}_{i,0}, \hat{z}_{i,1})$ for each $i \in [n]$. As in the previous case, define $x_i \in \{0,1\}$ so that t contains $\hat{z}_{i,x_i}$ for each $i \in [n]$. Now since the index set of $\hat{z}_{i,x_i}$ contains a factor of $X_{i,1-x_i}^{\deg(x_i)}$, and the top-level index set $\mathcal{U}$ contains only $\deg(x_i)$ copies of $X_{i,1-x_i}$, we conclude that for each $i \in [n]$, the monomial t does not contain any factors of $\hat{x}_{i,1-x_i}, \hat{u}_{i,1-x_i}$. In other words, it can be expressed as a monomial in the remaining encodings $(\hat{x}_{1,x_1}, \ldots, \hat{x}_{n,x_n}, \hat{u}_{1,x_1}, \ldots, \hat{u}_{n,x_n}, \hat{y}_j, \hat{v})$, satisfying case (2) of the lemma. □

Lemma A.5. *Fix an efficient adversary $\mathcal{A}$. For every valid zero-test polynomial z produced by $\mathcal{A}$ in the security game for Construction A.2 (in the generic model of composite-order multilinear maps), its input profile* $\mathrm{prof}(z)$ *is a set of strings in $\{0,1\}^n$, none of which is partial. Further, this set can be computed (albeit inefficiently) by examining z.*

Proof. The first part of the claim is immediate from Lemma A.4. To compute $\mathrm{prof}(z)$ (inefficiently), we can simply expand $\mathrm{prof}(z)$ into a sum of monomials and apply the definition of input profiles (Definition 3.13). □

Finally, we are ready to prove $i\mathcal{O}$ for Construction A.2.

Proof of Theorem A.3. As observed by Brakerski and Rothblum [BR14], $i\mathcal{O}$ is equivalent to a modified version of VBB (Definition 2.14) in which the simulator $\mathcal{S}$ is computationally unbounded. We will adopt this variant here, defining the simulator $\mathcal{S}$ exactly as in our proof of Theorem 3.5 (Section 3.5)—except that here, we invoke Lemma A.5 instead of Lemma 3.17, and thus our iteration over $\mathbf{x} \in \mathrm{prof}(z)$ may take up to exponential time. The rest of the proof follows the argument of Section 3.5. □

Performance analysis for $i\mathcal{O}$. Construction A.2 omits the $O(n^2)$ straddling-set elements necessary to achieve VBB (rather than $i\mathcal{O}$), and thus it improves on the parameters of the main construction (Construction 3.1). Specifically, using the "cross-multiplication" optimization (Section 4.2), the degree of multlinearity becomes $O(2^d + n)$, rather than $O(2^d + n^2)$; the obfuscation size becomes $O(m + n)$ ring elements, rather than $O(m + n^2)$; and the evaluation time becomes $O(s + n) = O(s)$, rather than $O(s + n^2)$. It is straightforward to adapt the other optimizations of Section 4.2 to the abridged Construction A.2, and in each case the $O(n^2)$ overhead is reduced to linear. Of course, the tradeoff is that by eliminating the extra elements, we only achieve $i\mathcal{O}$ rather than VBB. Constructing succinct (generic-model) VBB obfuscation with only linear overhead remains an interesting open problem.

B Formal Polynomials in the Generic Model

To prove security of our main construction (Construction 3.1), we use a more intuitive characterization of the generic multilinear map oracle: rather than queries in terms of "handles" (nonces),

as defined formally in Definition 2.10, we consider zero-test queries that refer to formal polynomials (Remark 2.11), whose formal variables are substituted with their joint value distribution from the real game. The following definitions make this language precise.

Definition B.1 (Formal Variables of Construction 3.1)**.** For a given (keyed) arithmetic circuit $C : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$, the formal variables of Construction 3.1 are the following variables:

$$(\hat{x}_{i,b}, \hat{u}_{i,b}, \hat{z}_{i,b}, \hat{w}_{i,b})_{i\in[n],b\in\{0,1\}} (\hat{y}_j)_{j\in[m]}, \hat{v}, \hat{C}^*, (\hat{s}_{i_1,i_2,b_1,b_2})_{b_1,b_2\in\{0,1\},i_1<i_2\in[n]} \ .$$

Definition B.2 (Formal Polynomials for Handles)**.** Fix an adversary $\mathcal{A}$, and consider the state of the generic multilinear map oracle (Definition 2.10) in the virtual black-box security game (Definition 2.14), when played with the obfuscator $\mathcal{O}$ corresponding to our main construction (Construction 3.1) for a given (keyed) arithmetic circuit $C : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$. In the first phase of this game, the oracle answers queries from $\mathcal{O}$, while in the second phase it answers queries from $\mathcal{A}$. Here, as in the construction, we write the modulus of the composite-order map as $N = N_{\text{ev}} N_{\text{chk}}$. We also abuse notation here to write $\mathsf{CM.Encode}(\mathsf{sp}, [a,b]_S)$ to mean $\mathsf{CM.Encode}(\mathsf{sp}, v, S)$ for the value v that is congruent to a modulo $\mathbb{Z}_{N_{\text{ev}}}$ and b modulo $\mathbb{Z}_{N_{\text{chk}}}$ (as determined by the Chinese Remainder Theorem).

Let p be a formal polynomial (with integer coefficients) over the variables of Construction 3.1 (Definition B.1). During either phase of the game, we say that a value h (either a handle or an integer) *refers to* p (at index set S) if either:

- For some $b \in \{0,1\}, i \in [n]$, the handle h is the result of running $\mathsf{CM.Encode}(\mathsf{sp}, [b, \alpha_i]_{X_{i,b}})$ during step 6 of the obfuscator (resp., $\mathsf{CM.Encode}(\mathsf{sp}, [1,1]_{X_{i,b}})$, $\mathsf{CM.Encode}(\mathsf{sp}, [y_j, \beta_j]_Y)$, etc.), and p is the formal variable $\hat{x}_{i,b}$ (resp., $\hat{u}_{i,b}$, $\hat{y}_j$, etc.).
- The handle h is the result of a query $\mathsf{CM.Add}(h_1, h_2)$ and p is the polynomial $p_1 + p_2$, where h_1 refers to p_1 (at S) and h_2 refers to p_2 (at S) at the time of the query.
- The handle h is the result of a query $\mathsf{CM.Mult}(h_1, h_2)$ and p is the polynomial $p_1 p_2$, where h_1 refers to p_1 (at S_1) and h_2 refers to p_2 (at S_2) at the time of the query; $S_1 \cap S_2 = \emptyset$; and $S = S_1 \cup S_2$.
- The value h is an integer $c \in \mathbb{Z}$; p is the constant polynomial c; and $S = \emptyset$.

Definition B.3 (Real Values for Formal Polynomials)**.** Fix an adversary $\mathcal{A}$, and consider the state of the generic multilinear map oracle during the security game (as formalized in Definition B.2).

Let p be a formal polynomial (with integer coefficients) over the variables of Construction 3.1 (Definition B.1). During either phase of the game, for distributions r, r_1, r_2 over $\mathbb{Z}_N$ (implicitly conditioning on N), we say that the *real value* of p is r if either:

- For some $b \in \{0,1\}, i \in [n]$, the polynomial p is the formal variable $\hat{x}_{i,b}$ (resp., $\hat{u}_{i,b}$, $\hat{y}_j$, etc.), and r is the distribution of the value of $[b, \alpha_i]_{X_{i,b}}$ generated in step 6 of the obfuscator (resp., $[1,1]_{X_{i,b}}$, $[y_j, \beta_j]_Y$, etc.).
- The polynomial p is formally written as $p_1 + p_2$ for some p_1, p_2; the real value of p_1 is r_1; the real value of p_2 is r_2; and $r = r_1 + r_2$.
- The polynomial p is formally written as $p_1 p_2$ for some p_1, p_2; the real value of p_1 is r_1; the real value of p_2 is r_2; and $r = r_1 r_2$.
- The polynomial p is formally written as c for a constant $c \in \mathbb{Z}$, and $r = c$.

Definition B.4 (Index Sets for Formal Polynomials)**.** *Fix an adversary $\mathcal{A}$, and consider the state of the generic multilinear map oracle during the security game (as formalized in Definition B.2).*

Let p be a formal polynomial (with integer coefficients) over the variables of Construction 3.1 (Definition B.1). We say that the index set of p is $S \subset \mathcal{U}$ if either:

- *The polynomial p is the formal variable $\hat{x}_{i,b}$ (resp., $\hat{u}_{i,b}$, $\hat{y}_j$, etc.), and S is $X_{i,b}$ (resp., $X_{i,b}$, Y, etc.).*
- *The polynomial p is formally written as $p_1 + p_2$ for some p_1, p_2; the index set of p_1 is S; and the index set of p_2 is S.*
- *The polynomial p is formally written as $p_1 p_2$ for some p_1, p_2; the index set of p_1 is S_1; the index set of p_2 is S_2; $S_1 \cup S_2 \subseteq \mathcal{U}$; and $S = S_1 \cup S_2$.*
- *The polynomial p is formally written as c for a constant $c \in \mathbb{Z}$, and $S = \emptyset$.*

As an immediate consequence of these definitions, we note the following intuitive properties:

Lemma B.5 (Uniqueness of Real Values)**.** *Fix formal polynomials p_1, p_2 (with integer coefficients) in the variables of Construction 3.1 (Definition B.1). During the security game (as formalized in Definition B.2), if the polynomials p_1 and p_2 are identically equal and the real value (Def. B.3) of p_1 is r, then the real value of p_2 is r.*

Proof. Since $p_1 \equiv p_2$, they are identical when expanded into a sum of monomials. Thus it suffices to prove the claim for a single distributive step. If the real value of $(p_1 + p_2)p_3$ is r, then by case analysis we conclude that that the real values of p_1, p_2, p_3 are, resp., some r_1, r_2, r_3 such that $(r_1 + r_2)r_3 = r$. Hence the real value of $p_1 p_3 + p_2 p_3$ is $r_1 r_3 + r_2 r_3$ as desired. □

Lemma B.6 (Uniqueness of Index Sets)**.** *Fix formal polynomials p_1, p_2 (with integer coefficients) in the variables of Construction 3.1 (Definition B.1). If the polynomials p_1 and p_2 are identically equal and the index set (Def. B.4) of p_1 is r, then the index set of p_2 is r.*

Proof. Since $p_1 \equiv p_2$, they are identical when expanded into a sum of monomials. Thus it suffices to prove the claim for a single distributive step. If the index set of $(p_1 + p_2)p_3$ is S, then by case analysis we conclude that that the index sets of p_1, p_2, p_3 are, resp., S_1, S_1, and S_3, for some sets S_1, S_3 such that $S_1 \cup S_3 = S$. Hence the index set of both $p_1 p_3$ and $p_2 p_3$ is $S_1 \cup S_3 = S$, and thus so is that of their formal sum $p_1 p_3 + p_2 p_3$, as desired. □

Lemma B.7 (Evaluation Commutes With Substutition)**.** *During the security game (as formalized in Definition B.2), in the generic graded encoding model, suppose a handle h is mapped, at index set S, to a value in the oracle's table whose (prior) distribution is r. Then h refers (Def. B.2) to a formal polynomial p whose index set (Def. B.4) is S and whose real value (Def. B.3) is r.*

Proof. By structural induction on the sequence of CM.Add, CM.Mult queries by which the mapping was formed. □

In our main proofs in Section 3, we make use of Lemmas B.5, B.6, B.7 implicitly. We also make implicit use of the fact that it is easy to compute the formal polynomial (and index set) that a handle refers to, given the sequence of queries and responses between the oracle and the machines $\mathcal{O}, \mathcal{A}$, simply by following the inductive definition (Definition B.2).