How to Obfuscate Programs Directly

Joe Zimmerman

Stanford University

Abstract. We propose a new way to obfuscate programs, via 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. With "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.
Our construction is efficient enough that if "clean" multilinear maps were known, then general-purpose program obfuscation could become implementable in practice. Our results demonstrate that the question of "clean" multilinear maps is not a technicality, but a central open problem.

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 security (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 works of Ananth et al. [AGIS14] and Sahai and Zhandry [SZ14] investigate the question of optimizing obfuscation, and obtain significant improvements for some specific cases, but much work remains to be done. One major source of inefficiency is that in all previously known constructions, including those of [AGIS14, SZ14], 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 (i.e., 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 (such as the CLT construction), or with "clean" maps, for which there is still no known candidate. 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 "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. All other known approaches require a number exponential in the circuit's depth, since every sub-circuit with fanout greater than 1 must be duplicated before converting the circuit to a matrix branching program.

Remark 1.1 (Cryptanalysis of CLT). For some time, it was believed that the CLT construction [CLT13], together with the modifications of [GLW14, App. B], provided a secure instantiation of a composite-order multilinear map. The obfuscation construction we develop in this work depends fundamentally on the composite-order setting, for which CLT has been the natural candidate instantiation.

However, subsequent to this work, there has been significant progress in the cryptanalysis of the CLT multilinear map. Generalizing the "zeroizing" attack of [GGH13a], Cheon, Han, Lee, Ryu, and Stehlé [CHL$^+$14] have shown that given public encodings of a certain form, it is possible to factor the CLT modulus and thereby break the scheme. Further cryptanalysis showed that it does not help to rule out encodings of this particular form; variants of the Cheon et al. attack can be executed under much weaker hypotheses [GHMS14, BWZ14, CLT14].

In light of these attacks, it is not clear that composite-order multilinear maps can currently be instantiated. We are optimistic that new candidates will be discovered in the future, perhaps as variants of other known constructions [GGH13a, GGH14], and we await the development of suitable multilinear maps.

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 $O(2^d + n^2)$.

	Degree of multilinearity	Obfuscation size (# ring elements)	Evaluation time (# ring operations)
Via Barrington [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 general (unkeyed) 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 multilinear map (Section 2.8). For "clean" maps (whose existence is still open), the cost is just poly(λ). For "noisy" maps, the cost depends on the instantiation; e.g., for the CLT map [CLT13], the reader should multiply each row's obfuscation size and evaluation time by $O(\deg^{2+\varepsilon'}) \cdot \mathrm{poly}(\lambda)$, where deg is the corresponding multilinearity degree from the first column, and ε' is a small constant determined by the choice of the Θ parameter in composite-order CLT [GLW14, App. B].

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

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

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.

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 (see the full version [Zim14] for details). 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 for general-purpose obfuscation [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

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

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 leave the full development for future 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 defer the full details of this extension to the full version [Zim14]. At a high level, 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). We remark that this extension is especially apt for algorithms such as block ciphers, which maintain and update

[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.

a small "working state" and read off a (multi-bit) output from that state at the end of the computation.

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 standard universal-program transformations; we defer the formal details to the full version [Zim14].)

Known approaches. All known constructions of general program obfuscation (including this work) are based on *multilinear maps* [BS03, GGH13a, CLT13, GGH14]. Multilinear maps, 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]. 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.

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

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

$\mathbb{Z}_N$ 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 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 is 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 Construction 1.)

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 constructing 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 $(\hat{x}_{1,0}, \hat{x}_{1,1}, \hat{x}_{2,0}, \hat{x}_{2,1}, \ldots)$ at its own singleton index set $(X_{1,0}, X_{1,1}, X_{2,0}, X_{2,1}, \ldots)$. 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

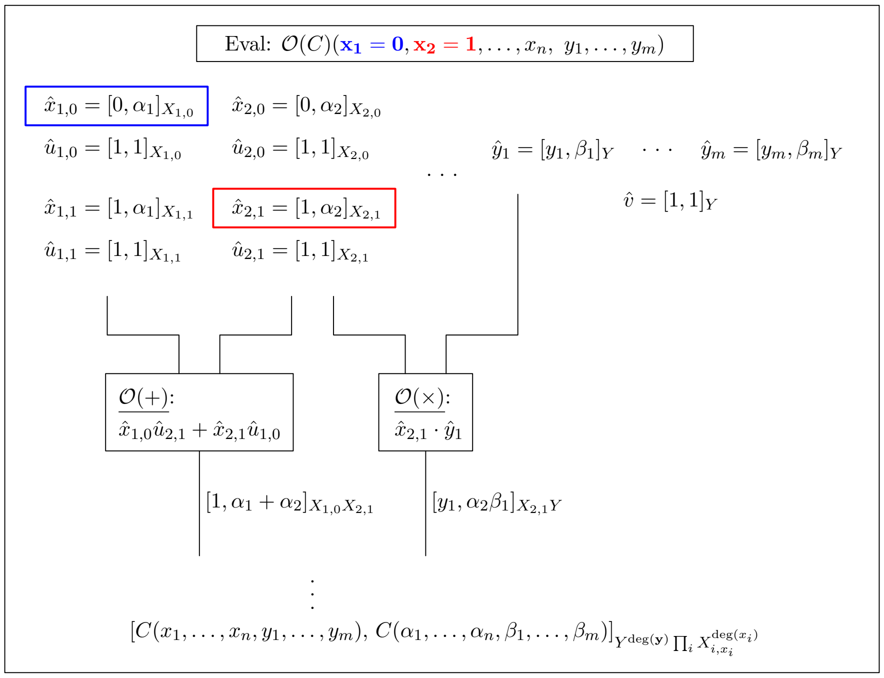
Eval: $\mathcal{O}(C)(\mathbf{x_1 = 0}, \mathbf{x_2 = 1}, \ldots, x_n,\ y_1, \ldots, y_m)$
$\hat{x}_{1,0} = [0, \alpha_1]_{X_{1,0}}$
$\hat{x}_{2,0} = [0, \alpha_2]_{X_{2,0}}$
$\hat{u}_{1,0} = [1, 1]_{X_{1,0}}$
$\hat{u}_{2,0} = [1, 1]_{X_{2,0}}$
$\hat{y}_1 = [y_1, \beta_1]_Y$
$\hat{y}_m = [y_m, \beta_m]_Y$
$\hat{x}_{1,1} = [1, \alpha_1]_{X_{1,1}}$
$\hat{x}_{2,1} = [1, \alpha_2]_{X_{2,1}}$
$\hat{v} = [1, 1]_Y$
$\hat{u}_{1,1} = [1, 1]_{X_{1,1}}$
$\hat{u}_{2,1} = [1, 1]_{X_{2,1}}$
$\mathcal{O}(+)$: $\hat{x}_{1,0}\hat{u}_{2,1} + \hat{x}_{2,1}\hat{u}_{1,0}$
$\mathcal{O}(\times)$: $\hat{x}_{2,1} \cdot \hat{y}_1$
$[1, \alpha_1 + \alpha_2]_{X_{1,0}X_{2,1}}$
$[y_1, \alpha_2\beta_1]_{X_{2,1}Y}$
$[C(x_1, \ldots, x_n, y_1, \ldots, y_m),\ C(\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_m)]_{Y^{\deg(\mathbf{y})} \prod_i X_{i,x_i}^{\deg(x_i)}}$

Fig. 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 $y_1, \ldots, y_m$.)

$\mathcal{U}$; but their index sets prevent the adversary from making any input-inconsistent choices within a given monomial.

Enforcing separability: componentwise blinding factors. Our "interlocking" elements $\hat{z}_{i,b}$ also contain additional blinding factors: $\delta_{i,b}$, in the evaluation component; and $\gamma_{i,b}$, in the "check" component (see Section 3 for details). Intuitively, we need the factors $\gamma_{i,b}$ to make sure that even if the adversary submits a "mixed" query that refers to more than one input string $\mathbf{x}$, the parts of the adversary's query that refer to different input strings are separated, each scaling a different γ monomial, so that they cannot cancel each other and thus the simulator can answer the zero-test query by addressing each consistent input independently.[6] The values $\delta_{i,b}$ are needed in the general case of arithmetic circuits (rather than just Boolean circuits), since there we may have integer outputs $C(\mathbf{x}, \mathbf{y}) \neq C(\mathbf{x}', \mathbf{y})$ such that neither $C(\mathbf{x}, \mathbf{y})$ nor $C(\mathbf{x}', \mathbf{y})$ is zero, and the adversary still should not be allowed to learn whether some specific linear combination of $C(\mathbf{x}, \mathbf{y})$ and $C(\mathbf{x}', \mathbf{y})$ is zero. Together with the design of the index sets described above, these blinding factors let us decompose the adversary's queries into independent subqueries each consistent with one input string $\mathbf{x} \in \{0,1\}^n$, which is essential for the construction of an efficient simulator in the VBB 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 Section 3.

[6] In this respect, the γ values in our construction play the same role as the scalar blinding factors in some other obfuscators (e.g., the α factors in [BGK$^+$14]).

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 show in the full version [Zim14, §4], 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.

In concurrent and independent work, Applebaum and Brakerski [AB15] describe an obfuscator that is very similar to the simplified $i\mathcal{O}$ version of our construction [Zim14, Appendix A]. The construction of Applebaum and Brakerski only achieves the weaker notion of (generic-model) $i\mathcal{O}$, rather than (generic-model) VBB security as we achieve in our main construction [Zim14, §3]. Applebaum and Brakerski also give an extension that provides robustness in the stronger setting of low-level zero-test operations (see Remark 2.6 below), at the cost of n additional components in the composite-order encodings. We note that our construction in this work can also be extended to be robust in this stronger setting, at the cost of only 2 additional components, via the generic transformation of [BWZ14]. (Indeed, the zero-immunizing transformation of [BWZ14] applies to any scheme secure in the generic model of composite-order multilinear maps, and thus also improves the construction of Applebaum and Brakerski.)

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 $\mathsf{Uniform}(\mathsf{Primes}[a, b])$. Following standard conventions of cryptography, we also 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 λ.

2.2 Oblivious Computation and the "Mux" Operation

A program is considered *data-oblivious*, or *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. 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 "$\text{if } x \text{ then } y \leftarrow z;\ \text{else } y \leftarrow w;$", 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 "$y \leftarrow \mathrm{mux}(x, z, w)$".

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. 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. We note that straight-line programs are naturally identified with arithmetic circuits (fanin two, unbounded fanout). In this work, we will view straight-line programs and arithmetic circuits interchangeably.

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,[7] provided that the machine is oblivious (Section 2.2). We also note that an arithmetic circuit $C : \{0,1\}^n \to \{0,1\}$ can be expressed as a formal multivariate polynomial in $\mathbb{Z}[x_1, \ldots, x_n]$ (perhaps after duplicating gates to account for fanout), and we will identify circuits with their corresponding polynomials. Although the multivariate polynomial for a given circuit C may be of exponential size, it can still be evaluated efficiently, and we can perform algebraic substitutions 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. 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 the formal details, we refer the reader to the full version [Zim14].

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 2.1 (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 make use of standard transformations from general circuit families to keyed circuit families, in which the secret key is the entire circuit to be obfuscated, and C is a universal circuit; we defer the formal details to the full version [Zim14]. We emphasize that these universal-circuit transformations are mainly for theoretical purposes. In practice, a much better approach would be to design, for each desired cryptographic application of obfuscation, a family of circuits that is already keyed with respect to the particular secret that needs to be hidden.

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

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 $\hat{x}, \hat{y}$. 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 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, 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 2.2 (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 2.3 (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 2.4 (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 2.3), 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 components to generate for the modulus. It produces public parameters pp, secret parameters sp, and integers $N_1, \dots, N_k$ as follows:

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

Each integer $N_1, \dots, N_k$ 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 $\mathcal{S}$:

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

– 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 $\mathcal{U}$:

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

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

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

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

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

– 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.

[8] We remark here that our construction does not rely on the individual moduli $N_1, \dots, N_k$ 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.

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 2.5 (Notation for Encodings of Direct Products). We use the notation $[x_1, x_2, \ldots, x_k]_S$ 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 generic model is similar to the generic group model [Sho97]: intuitively, in the generic model, the only thing an adversary can do with ring elements is to apply the multilinear map operations.

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 and zero-tests internally). For the complete exposition and formal definitions, we refer the reader to the full version [Zim14].

Remark 2.6 (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 candidate multilinear maps based on noisy encodings (e.g., [GGH13a, GGH14]), 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. For such a modification, we refer the reader to the work of Boneh, Wu, and Zimmerman [BWZ14, §3], in which the authors show a generic transformation for composite-order multilinear maps that prevents the adversary from constructing nontrivial encodings of zero outside the top-level index set $\mathcal{U}$.

2.7 "Noisy" and "Clean" Multilinear Maps

The abstract definition of multilinear maps (Definition 2.4) 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 2.4, 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 2.7 ("Noisy" Composite-Order Multilinear Map). A *noisy composite-order multilinear map* is defined as in Definition 2.4, except that the top-level index set $\mathcal{U}$ has its multiplicities presented in unary.

We note that Definition 2.7 considers only the noise growth due to multiplication operations, and disregards that of addition operations.[10] 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 to multilinear maps that may not incorporate noise terms in the same way as the approaches currently known.

[10] 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.

2.8 Instantiation of Composite-Order Multilinear Maps

As discussed above in Remark 1.1, until very recently it was believed that the CLT scheme [CLT13] provided a secure instantiation of "noisy" composite-order multilinear maps. For completeness, we now briefly recount the structure of the CLT 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. 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. For a more comprehensive exposition, we refer the reader to the full version [Zim14].

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 along these lines. For the full analysis, we refer the reader to [GLW14, Appendix B].

Possible approaches for clean maps. While we know of some candidate strategies to instantiate "noisy" multilinear maps, 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, as would be required for "clean" maps. However, the current approach via encodings with random noise is not necessarily the only possible route. The theory of bilinear maps [Mil04, MOV93, Jou00, BF01] does not incorporate noise terms 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, 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 2.8 (Virtual Black-Box Obfuscation in an 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 extend Definition 2.8 in the standard way to entire circuit classes such as NC^1 and P/poly; we defer the formal details to the full version [Zim14]. We also note that since we require the obfuscator $\mathcal{O}$ to be efficient, the output of $\mathcal{O}$ is a circuit of size poly(λ), and thus the *polynomial slowdown* property of [BGI$^+$01] is immediate from the definition.

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 2.9 (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
$$\Pr[\tilde{C}_{\mathbf{y}} \leftarrow \mathcal{O}^{\mathcal{M}}(C, \mathbf{y}) \ ; \ \mathcal{O}.\mathrm{Eval}^{\mathcal{M}}(\tilde{C}_{\mathbf{y}}, C, \mathbf{x}) \neq C(\mathbf{x}, \mathbf{y})] \ < \ \varepsilon(\lambda) \ ,$$
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 2.10 (Succinct Virtual Black-Box Obfuscation). The definition is the same as Definition 2.9, 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 extend Definition 2.10 in the standard way to the classes P/poly and NC^1, just as in Definition 2.8.

2.11 Straddling Sets

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

An 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 the following slightly stronger property.

Definition 2.11 (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 $\mathcal{S}$, $\mathcal{S}_0 = (S_{0,1}, \ldots, S_{0,n})$ and $\mathcal{S}_1 = (S_{1,1}, \ldots, S_{1,n})$ 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 $\mathcal{T} = \mathcal{S}$.

We note that the simple construction of straddling sets in [BGK+14] already satisfies our stronger definition. We defer the details to the full version [Zim14].

3 Construction

We now present our main obfuscation construction (Construction 1), which acts on keyed circuits (Section 2.4) as depicted in Figure 1. (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 1 (Construction of 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 2.4). 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 (Definition 2.11) 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 $\mathsf{BitFill}_{i_1,i_2,b_1,b_2} = S_{i_1,b_1,i_2} S_{i_2,b_2,i_1}$.
2. Construct the following index set of fresh formal symbols 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 $(\mathsf{CM.pp}, \mathsf{CM.sp}, N_{\text{ev}}, N_{\text{chk}}) \leftarrow \mathsf{CM.Setup}(\mathcal{U}, 1^{d+\lambda}, 2)$, indicating a security parameter of $d + \lambda$ for the multilinear map, and a modulus that decomposes into two factors $N = N_{\text{ev}} N_{\text{chk}}$.
4. For each $i \in [n]$, generate uniformly random values $\alpha_i, \gamma_{i,0}, \gamma_{i,1} \leftarrow \mathbb{Z}^*_{N_{\text{chk}}}$ and $\delta_{i,0}, \delta_{i,1} \leftarrow \mathbb{Z}^*_{N_{\text{ev}}}$. For each $j \in [m]$, generate a uniformly random value $\beta_j \leftarrow \mathbb{Z}^*_{N_{\text{chk}}}$.
5. Compute the check value $C^* = C(\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_m) \in \mathbb{Z}_{N_{\text{chk}}}$.
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 2.5):
$$\hat{x}_{i,b} = [b, \alpha_i]_{X_{i,b}} \qquad \hat{u}_{i,b} = [1,1]_{X_{i,b}} \qquad \hat{y}_j = [y_j, \beta_j]_Y \qquad \hat{v} = [1,1]_Y$$
$$\hat{z}_{i,b} = [\delta_{i,b}, \gamma_{i,b}]_{X_{i,1-b}^{\deg(x_i)} Z_i W_i \, \mathsf{BitCommit}_{i,b}} \qquad \hat{w}_{i,b} = [0, \gamma_{i,b}]_{W_i \, \mathsf{BitCommit}_{i,b}}$$
$$\hat{C}^* = [0,\ C^*]_{Y^{\deg(\mathbf{y})} \prod_{i\in[n]} (X_{i,0}X_{i,1})^{\deg(x_i)} Z_i}$$
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 2.5):
$$\hat{s}_{i_1,i_2,b_1,b_2} = [1,1]_{\mathsf{BitFill}_{i_1,i_2,b_1,b_2}}$$
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:

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

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 $\mathcal{O}.\text{Eval}(\tilde{C}_{\mathbf{y}}, C, \mathbf{x})$ operates as follows.

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

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 1, 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 raise the index set to $S \cup T$ for both resulting elements. Then, using $\mathsf{CM.Add}$, output the sum of the two elements.

   We note that the result of this procedure, for each sub-circuit of C, will be an encoding whose index set consists of 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:

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

   (We remark that while we present simple algorithms here for clarity, there are many natural optimizations; for details, we refer the reader to the full version [Zim14, §4].)
2. Using the procedures $\mathsf{CM.Add}$, $\mathsf{CM.Mult}$, compute the unit encoding $\hat{\sigma} = \prod_{i_1<i_2\in[n]} \hat{s}_{i_1,i_2,x_{i_1},x_{i_2}}$, and compute the following encoded element:

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

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

The correctness of Construction 1 is straightforward from the definitions of the multilinear map operations, and we defer the proof to the full version [Zim14].

Succinctness. In Construction 1, we instantiate the multilinear map with a security parameter of $d + \lambda$, rather than λ. As detailed in the full version [Zim14], 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 2.10), 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, by using a *computational* analog of the Schwartz-Zippel lemma (generalizing an elegant result of Boneh and Lipton [BL96]). We defer the details of this modification and its proof to the full version [Zim14]; here we just state the modified ("succinct") version of the construction.

Construction 2 (Virtual Black-Box Obfuscation (Succinct Version)). Proceed as in Construction 1, except in step 3, provide 1^λ as the security parameter to $\mathsf{CM.Setup}$, rather than $1^{d+\lambda}$.

Remark 3.1 (Indistinguishability Obfuscation). Our main result shows that Construction 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; notably, we avoid the $O(n^2)$ cost of the straddling-set encodings. For continuity, we defer the details of this modification to the full version [Zim14, 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. For space reasons, we defer the proofs of the main theorems to the full version [Zim14].

Our construction can be based either on "noisy" or on "clean" multilinear maps. Since we operate on 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, 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]. Moreover, 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 2) achieves *succinct* VBB obfuscation (Definition 2.10) for P/poly.

Theorem 3.2. *Suppose that factoring is hard on average. Then Construction 2 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 3.2, since there we do not assume the hardness of factoring.[11]

Theorem 3.3. *Construction 1 (composed with a universal circuit simulation) achieves virtual black-box obfuscation for P/poly in the generic model of* clean *composite-order multilinear maps.*

Of course, it is still unknown how one might construct "clean" multilinear maps, and thus we prove separately that we achieve obfuscation for NC^1 given only "noisy" maps. 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 3.4. *Construction 1 (composed with a universal circuit simulation) 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.

4 Performance Analysis

We now discuss the asymptotic efficiency of our main construction. We give only a very brief summary here; for more details, we encourage the reader to follow the exposition in the full version [Zim14, §4].

First, we establish the basic performance parameters. It turns out that the time to evaluate an obfuscated circuit is dominated by the "raising" operations for addition gates, in which we multiply elements by unit encodings in order to make the index sets match. This fact dictates the overall optimization strategy; in the full version [Zim14] we give the details of two approaches ("cross-multiplication" and "pre-mixing"), which reduce this evaluation cost with different tradeoffs. Using the "cross-multiplication" optimization, for a keyed arithmetic circuit $C : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$ of size s and depth d, we find that the multilinearity degree required is $O(2^d + n^2)$, the (keyed) obfuscation size is

[11] We remark that this distinction is nontrivial: as far as we know, the existence of composite-order multilinear maps does not necessitate the hardness of factoring, even though the concrete instantiation via the CLT scheme [CLT13] would be trivially broken if factoring were easy.

$O(m+n^2)$ ring elements, and the evaluation time is $O(s+n^2)$ ring operations.[12] In particular, the number of ring elements depends only on the secret part of the circuit, i.e., the key $\mathbf{y} \in \{0,1\}^m$. Moreover, excluding the $O(n^2)$ operations that arise from straddling sets, the number of ring operations is proportional to the circuit size—reflecting the fact that our evaluation algorithm follows C's structure directly.

We also specialize our performance analysis to standard settings (both keyed and unkeyed), to provide a more direct comparison with other known approaches. For instance, for balanced Boolean formulas (unkeyed NC^1 circuits of depth d, with input length n bits), the multilinearity degree is only $\Theta(2^d n + n^2)$, and we require only $\Theta(2^d n + n^2)$ ring elements and operations—as compared with the standard approach via Barrington's theorem [GGH$^+$13b, BR14, BGK$^+$14], for which all three metrics are $\Theta(4^d n + n^2)$; or the parameterized approach of [AGIS14, Gie01], for which the degree is $\Theta(2^{(1+\varepsilon)d} n + n^2)$ and the other two parameters are $\Theta(2^{(1+\varepsilon)d} 4^{2/\varepsilon} n + n^2)$.

More generally, since our new obfuscator's evaluation mirrors the structure of the original circuit, we find that our techniques expose a rich new design space of algorithms that can be *input* to the obfuscator. For example, to specialize our 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 (as detailed in the full version [Zim14]). We feel that these applications are only the beginning, and we hope that this work will encourage further study of obfuscating *specific*, keyed circuit families.

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

[12] We present the cost here in terms of ring elements and ring operations. The concrete cost in bits and bit operations depends on the multilinear map (Section 2.8). For "clean" maps (whose existence is still open), the cost is just poly(λ). For "noisy" maps, the cost depends on the instantiation; e.g., for the CLT map [CLT13], the reader should multiply every obfuscation size and evaluation time by $O(\deg^{2+\varepsilon'}) \cdot \mathrm{poly}(\lambda)$, where deg is the multilinearity degree required, and ε' is a small constant determined by the choice of the Θ parameter in composite-order CLT [GLW14, App. B].

(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 [GGH13a], 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.

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$^+$01]. Moreover, our results in this work highlight the fact that there are simple concrete examples of differences between the generic model and its instantiation via the CLT scheme—for instance, in one optimization based on the Fast Fourier Transform (detailed in the full version [Zim14, §4]), our computation is valid for CLT encodings but cannot be implemented in the generic model. While this particular difference is fortuitous, we are led to consider whether there are other algebraic properties that hold in the CLT scheme—and may, in fact, be compatible with concrete security assumptions, such as that of [GLW14]—yet which may indicate fundamental weaknesses in the generic model as it is used here and in [GGH$^+$13b, BR14, BGK$^+$14]. 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 is efficient enough that if "clean" multilinear maps were known, then general-purpose obfuscation could become implementable in practice. 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.

Bibliography

[AB15] Benny Applebaum and Zvika Brakerski. Obfuscating circuits via composite-order graded encoding. Cryptology ePrint Archive, Report 2015/025, 2015. http://eprint.iacr.org/.

[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/.

[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.

[BL96] Dan Boneh and Richard J. Lipton. Algorithms for black-box fields and their application to cryptography (extended abstract). In *CRYPTO*, 1996.

[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.

[BWZ14] Dan Boneh, David J. Wu, and Joe Zimmerman. Immunizing multilinear maps against zeroizing attacks. Cryptology ePrint Archive, Report 2014/930, 2014. http://eprint.iacr.org/.

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

[CHL$^+$14] Jung Hee Cheon, Kyoohyung Han, Changmin Lee, Hansol Ryu, and Damien Stehle. Cryptanalysis of the multilinear map over the integers. Cryptology ePrint Archive, Report 2014/906, 2014. http://eprint.iacr.org/.

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

[CLT14] Jean-Sebastien Coron, Tancrede Lepoint, and Mehdi Tibouchi. Cryptanalysis of two candidate fixes of multilinear maps over the integers. Cryptology ePrint Archive, Report 2014/975, 2014. http://eprint.iacr.org/.

[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.

[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/.

[GHMS14] Craig Gentry, Shai Halevi, Hemanta K. Maji, and Amit Sahai. Zeroizing without zeroes: Cryptanalyzing multilinear maps without encodings of zero. Cryptology ePrint Archive, Report 2014/929, 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.

[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.

[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.

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

[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.

[SZ14] Amit Sahai and Mark Zhandry. Obfuscating low-rank matrix branching programs. Cryptology ePrint Archive, Report 2014/773, 2014. http://eprint.iacr.org/.

[Zim14] Joe Zimmerman. How to obfuscate programs directly. Cryptology ePrint Archive, Report 2014/776, 2014. http://eprint.iacr.org/.