Obfuscating Circuits via Composite-Order Graded Encoding[*]

Benny Applebaum[†] Zvika Brakerski[‡]

Abstract

We present a candidate obfuscator based on composite-order Graded Encoding Schemes (GES), which are a generalization of multilinear maps. Our obfuscator operates on circuits directly without converting them into formulas or branching programs as was done in previous solutions. As a result, the time and size complexity of the obfuscated program, measured by the number of GES elements, is directly proportional to the circuit complexity of the program being obfuscated. This improves upon previous constructions whose complexity was related to the formula or branching program size. Known instantiations of Graded Encoding Schemes allow us to obfuscate circuit classes of polynomial degree, which include for example families of circuits of logarithmic depth.

We prove that our obfuscator is secure against a class of generic algebraic attacks, formulated by a generic graded encoding model. We further consider a more robust model which provides more power to the adversary and extend our results to this setting as well.

As a secondary contribution, we define a new simple notion of *algebraic security* (which was implicit in previous works) and show that it captures standard security relative to an ideal GES oracle.

1 Introduction

General-purpose program obfuscation allows us to transform an arbitrary computer program into an "unintelligible" form while preserving its functionality. Syntactically, an obfuscator for a function family $\mathcal{C} = \{C_K\}$ is a randomized algorithm that maps a function $C_K \in \mathcal{C}$ (represented by an identifier K) into a "program" $\hat{C} \in \{0,1\}^*$. The obfuscated program should preserve the functionality of C_K while hiding all other information about C_K. The first property is formalized via the existence of an efficient evaluation algorithm Eval which, given an input x and an obfuscated program $\hat{C}$, outputs $C_K(x)$. The second property has several different formulations, most notably *Virtual Black-Box* (VBB) and *Indistinguishability Obfuscation* (iO) [BGI+12].

The first candidate general-purpose obfuscator has been introduced by Garg et al. [GGH+13b]. Their work and follow-ups such as [BR14b, BGK+14] relied on *Graded Encoding Schemes* (GES) [GGH13a, CLT13] which generalize the more traditional notion of multilinear maps. All these

[*]A preliminary version of this paper appears in the proceedings of TCC 2015.

[†]Tel-Aviv University, `bennyap@post.tau.ac.il`. Supported by the European Unions Horizon 2020 Programme (ERC-StG-2014-2020) under grant agreement no. 639813 ERC-CLC, ISF grant 1155/11, Israel Ministry of Science and Technology (grant 3-9094), GIF grant 1152/2011, and the Check Point Institute for Information Security.

[‡]Weizmann Institute of Science, `zvika.brakerski@weizmann.ac.il`. Supported by the Israel Science Foundation (Grant No. 468/14), Binational Science Foundation (Grants No. 2016726, 2014276), and by the European Union Horizon 2020 Research and Innovation Program via ERC Project REACT (Grant 756482) and via Project PROMETHEUS (Grant 780701).

works share a similar two-step outline. First it is shown how to use the GES to obfuscate function families from a weak complexity class such as $\mathcal{NC}^1$ (the class of polynomial-size circuits with logarithmic depth and bounded fan-in gates), and then the weak obfuscator is bootstrapped into a general-purpose obfuscator for arbitrary polynomial-size circuits based on low-complexity fully homomorphic encryption [GGH$^+$13b, BR14b, BGK$^+$14] or on low-complexity pseudorandom functions [GIS$^+$10, App14]. Following [AGIS14], we refer to the first step as the "core obfuscator".

So far, all known core obfuscators are applied to the branching program representation of the function. Hence, in order to obfuscate some family of circuits (say in $\mathcal{NC}^1$) one has to first convert the given circuit into a branching program, and only then use the core obfuscator. This is both unnatural and inefficient. Indeed, the complexity of existing core obfuscators (in terms of computation, program size, and number of multilinear levels) grows with the *formula size* or *branching program* size of the obfuscated program, which are polynomially larger than the *circuit size*. (See Section 1.3 for quantitative comparison.) From a more principal point of view, one may wonder whether the use of branching programs is inherent or is just a limitation of our current techniques. In this paper, we study the existence of "direct circuit obfuscators". Specifically, we ask:

> Is it possible to construct a function family $\mathcal{C}$ by a core obfuscator whose complexity is linear in the *circuit complexity* of $\mathcal{C}$?

Following [AGIS14], we assume that the family $\mathcal{C} = \{C_K\}$ is represented by some universal evaluator $\mathcal{U}$ which given an index K of a function $C_K \in \mathcal{C}$ and an input x outputs the value $C_K(x)$. The (circuit) complexity of $\mathcal{C}$ is measured by the (circuit) complexity of $\mathcal{U}$ and the size of $\mathcal{C}$ is measured by the bit-length of the identifiers K. (These are natural complexity measures which lower-bound the time/size complexity of any obfuscator for $\mathcal{C}$. See Remark 1.2.)

1.1 Our Results

We take a step towards answering the above question in the affirmative: We construct new core obfuscators for any circuit family $\mathcal{C}$, where the size of an obfuscated program, measured by the number of GES elements, is proportional to the size of $\mathcal{C}$, and its time-complexity, measured by the number of GES operations, is proportional to the circuit complexity of $\mathcal{C}$.

When instantiating with current GES candidates, this falls short of a full answer to the above question since in these instantiations the element size depends on the total evaluated *degree*, a property which is inherited by our constructions. Our constructions are based on composite order Graded Encoding Scheme [GLW14], and are proved to achieve indistinguishability against adversaries which are limited to algebraic attacks allowed in a generic GES model. In fact, we study two different variants of the generic model (one is weaker than the other) and provide corresponding constructions in each of these models. Before stating our results, a few words about generic models are in order.

Generic Graded Encoding Schemes. A *Graded Encoding Scheme* is parameterized by a ring $\mathcal{R}$ and a top level multiset $\mathbf{v}_{\mathrm{zt}}$ over the universe $[\tau]$. Intuitively, the GES defines (exponentially) many encodings of the ring where each encoding is indexed by a multiset $\mathbf{v} \subseteq \mathbf{v}_{\mathrm{zt}}$. (A multiset is represented by an integer vector $\mathbb{N}^\tau$.) In our first (and weaker) generic model $\mathcal{MRG}$ (for *Multiple-encoding Random GES*), the encoding of a ring element $g \in \mathcal{R}$ under an index $\mathbf{v}$, denoted by $[g]_{\mathbf{v}}$, is distributed uniformly over an exponentially large set of random strings.[1] As a result, the adversary can manipulate encodings only via the use of a GES oracle that supports some restricted set of "legal" operations. In particular, the adversary is allowed to: (1) compute addition/subtraction for encodings that share the same index $[g]_{\mathbf{v}} \pm [g']_{\mathbf{v}} = [g \pm g']_{\mathbf{v}}$; (2) multiply elements with possibly distinct indices as long as the union of their indices is still a subset of $\mathbf{v}_{\text{zt}}$, i.e., $[g]_{\mathbf{v}} \times [g']_{\mathbf{v}'} = [g \cdot g']_{\mathbf{v} \bigcup \mathbf{v}'}$; and (3) zero-test a top-level encoding, i.e., $\mathsf{isZero}([g]_{\mathbf{v}_{\text{zt}}}) = 1 \Leftrightarrow g = 0$. The L_1 norm of $\mathbf{v}_{\text{zt}}$ (denoted $\|\mathbf{v}_{\text{zt}}\|_1$) upper bounds the total degree of polynomials that can be evaluated on the GES elements. In this model we prove the following theorem:

Theorem 1.1. *There exists an indistinguishability obfuscator* $\mathsf{SimpleObf}$ *with respect to $\mathcal{MRG}$ for any circuit family as follows. For a function family $\mathcal{C} = \{C_K\}_{K \in \{0,1\}^m}$ which is indexed by m-bit strings, operates on n-bit inputs, and can be universally computed by a t-size circuit of depth d, the obfuscation has the following properties:*

- *(Size) The description of the obfuscated program contains only $4n + 2m + 2$ encoded ring elements.*
- *(Evaluation complexity) The complexity of the evaluation algorithm is $O(t)$. In particular, the obfuscated program can be evaluated by an $O(t)$-size arithmetic circuit over the underlying GES ring with oracle gates to the GES oracle.*
- *(GES parameters) The underlying ring is a composite order ring $\mathbb{Z}_{p_1} \times \mathbb{Z}_{p_2}$ where p_1 and p_2 are co-primes and $\|\mathbf{v}_{zt}\|_1 \leq 2^d$.*

Remark 1.2. *The above parameters are essentially optimal (up to the dependency in the security parameter and in the encoding-size of a ring element under the GES oracle). Indeed, by the correctness property, any obfuscation scheme for $\mathcal{C}$ with size M and evaluation complexity T naturally defines an M-bit length indexing for $\mathcal{C}$ and a corresponding T-time universal evaluation algorithm. Hence, the size/time-complexity of the obfuscated program cannot beat the size/complexity of $\mathcal{C}$.*

Remark 1.3. *Our security proof requires certain properties on the underlying ring which in particular imply that the bit-lengths of the primes p_1, p_2 have to grow linearly with the depth d (and also with the security parameter), see formal statement in Theorem 5.3. As a result, assuming ideal $\mathcal{MRG}$ oracle, the bit length of an encoded ring element grows with d as well. Such ideal $\mathcal{MRG}$ oracle would still allow to instantiate our obfuscator for any class of polynomial size circuit with a-priori bounded polynomial depth.*

In known instantiations of GES, the bit length of encoded elements in fact grows with $\|\mathbf{v}_{zt}\|_1 \approx 2^d$ (due to nature of noise growth) which implies that we can only obfuscate circuits of logarithmic depth.

While the above scheme has a fairly low overhead, it relies on a strong generic model. Specifically, the scheme becomes insecure if the adversary manages to zero-test element whose encoding $[g]_{\mathbf{v}}$ lies in a low-level $\mathbf{v} \subsetneq \mathbf{v}_{\text{zt}}$. This vulnerability puts a strong restriction on the class of potential implementations. For example, GES in which each ring element has a unique encoding (in

[1] An alternative way to model a generic attack is to assume that the adversaries is given abstract handles to the encodings. We prefer the current model due to its simplicity and for the sake of consistency with previous works. We futrher note that security in the current (random string) model immediately implies security in the "handle model". Since our model only gives more power to the adversary and not to the honest user (since correctness should hold even for non random GES instantiations).

each level) cannot be used as it trivially permits low-level zero-testing.[2] Note that known candidates for *bilinear* maps have exactly this property. Currently, only few candidates for GES are known [GGH13a, CLT13], and based on our current understanding, it is possible to tweak these candidates into forbidding low-level zero-testing.[3] Still, it is desirable to obtain results in a more robust model which allows the adversary to zero-test low-level encodings. Formally, we define a different ideal GES oracle $\mathcal{URG}$ (for *Unique-encoding Random GES*) which, for every level $\mathbf{v}$, assigns for every ring element $g \in \mathcal{R}$ a *unique* (randomly chosen) encoding. Some works in the literature such as [BR14b] use models that are essentially $\mathcal{URG}$, while others such as [AGIS14] only provide proof in a weaker model that is similar to our $\mathcal{MRG}$. Our result in the $\mathcal{URG}$ model is as follows.[4]

Theorem 1.4. *There exists an indistinguishability obfuscator* RobustObf *with respect to* $\mathcal{URG}$ *for any circuit family* $\mathcal{C}$. *The parameters are similar to those of Theorem 1.1, except that the oracle* $\mathcal{URG}$ *needs to be defined over an* $(n+2)$*-composite order ring* $\mathcal{R} = \mathbb{Z}_{p_1} \times \cdots \times \mathbb{Z}_{p_{n+2}}$ *where* n *is the length of the input of the obfuscated program.*

The use of $O(n)$-composite order rings introduces an indirect efficiency overhead. Indeed, the description length of ring elements and the computational cost of oracle operations now grow with the input length n (and the security parameter). It is important to note that this overhead is independent of the size/complexity of $\mathcal{C}$, and so, for sufficiently large/complicated circuit families, we may still get an asymptotic advantage over alternative branching-program based approaches. We further note that Remark 1.3 applies to this case as well.

Definitional contribution. As already mentioned, we prove security relative to an ideal GES oracle (either $\mathcal{URG}$ or $\mathcal{MRG}$). Towards this goal, we propose a new algebraic abstraction of *canonical GES-based obfuscators* and define a corresponding notion of *algebraic security* (which was implicit in [BR14b, BGK+14, AGIS14, PST14]). Roughly speaking, a GES-based obfuscator is in canonical form, if given a program identifier K, it samples a tuple $a = (a_1, \dots, a_\ell)$ from the underlying ring $\mathcal{R}$, and then outputs a GES-encoding of these elements under the labels $(\mathbf{v}_1, \dots, \mathbf{v}_\ell)$ which depend only on the length parameters but otherwise are independent of the program identifer. Hence, an obfuscator is essentially a mapping from a program identifier K to distribution D_K over $\mathcal{R}^\ell$.

For algebraic security, an adversary A is a polynomial over ℓ variables (described by an arithmetic circuit) which is evaluated over the tuple $a = (a_1, \dots, a_\ell)$ sampled from D_K. The outcome of the attack is the bit $\mathsf{isZero}(A(a))$. Security requires the existence of a simulator S that given an oracle access to C_K can predict $\mathsf{isZero}(A(D_K))$ with all but negligible probability.

[2] To test if $[g]_\mathbf{v}$ is an encoding of zero simply check if the string $[g]_\mathbf{v} + [g]_\mathbf{v}$ equals to $[g]_\mathbf{v}$.

[3] It may be surprising that such tweaking could exist, since one can always increase the level from $\mathbf{v}$ to $\mathbf{v}_{\mathrm{zt}}$ by multiplying with a non-zero element of level $(\mathbf{v}_{\mathrm{zt}} - \mathbf{v})$. However, in known instantiations, it is possible to generate public parameters that cannot be used for generating new encodings, thus restricting the adversary to only have access to those levels provided in the obfuscated program. Those, in turn, are designed so there is no way to obtain an encoding at level $(\mathbf{v}_{\mathrm{zt}} - \mathbf{v})$ for "dangerous" values of $\mathbf{v}$. (This is somewhat similar to the "straddling sets" technique [BGK+14].)

[4] It is important to emphasize that the honest parties (the obfuscator and evaluator) do not exploit the fact that elements have unique encodings, as they are required to work with respect to any GES implementation. (See Section 3). Therefore, the $\mathcal{URG}$ model is strictly better than the $\mathcal{MRG}$ model in the sense that an obfuscation which is secure relatively to $\mathcal{URG}$ is also secure relatively to $\mathcal{MRG}$. The reverse direction does not necessarily hold as demonstrated by SimpleObf from Theorem 1.1.

Abstracting previous works, we show that algebraic security implies (standard) security relative to an ideal GES oracles.[5] Note that algebraic security considers only a static attack (a single query to D_K) whereas standard security (VBB or iO) corresponds to an adaptive game with possibly many GES queries for arithmetic/zero-testing operations. Roughly speaking, the proof maps each GES-query to a formal polynomial over some set of atomic variables; In order to simulate the answer of this query, it suffices to check whether the corresponding polynomial equals to one of the former polynomials. This can be done by using the algebraic-security simulator to zero-test the difference-polynomial. (See Lemma 3.7.)

Naturally, algebraic security is much easier to work with. (Indeed, a notable part of the proofs of [BR14b, BGK$^+$14] is devoted to essentially reducing standard security to algebraic security.) More importantly, algebraic security *does not* depend on the GES oracle at all. As such, it crystalizes the information-theoretic properties that are required in order to achieve security in an ideal GES model. We hope that this abstraction will be valuable for future constructions, and that our general lemma (algebraic security $\Rightarrow$ standard security) will allow to work directly with algebraic security.

1.2 Our Techniques

Obfuscation over locked boxes. To illustrate our techniques let us consider the following simplified physical model. The obfuscator is allowed to put ring elements in locked boxes that are labeled by multisets, such that everyone can add/multiply boxes at most T times (while respecting the aforementioned graded encoding multiset rules). After performing T operations, the box is opened only if its content is equal to zero.

A naive way to obfuscate a function C_K in this model is to put the identifier bits $K_1, \ldots, K_m$ in m separate boxes (labeled by $\mathbf{v}_1, \ldots, \mathbf{v}_m$ and referred to as "identifer" encodings), and prepare for each input x_i a pair of boxes (labeled by $\mathbf{v}_{i,0}, \mathbf{v}_{i,1}$ and referred to as input encoders) with the values 0 and 1. Given these boxes and an input x, the evaluator can choose the boxes $\mathbf{v}_{i,x_i}$, and propagate the values according to the circuit $\hat{\mathcal{U}}(\cdot,\cdot)$ which is the arithmetized version of the universal evaluator $\mathcal{U}$. As a result, we obtain a box that holds the value $\hat{\mathcal{U}}(x, K)$. Assuming that the circuit's size is T, the resulting box will be opened if and only if $C_K(x) = 0$.

This construction is insecure for several reasons. First and foremost, one can ignore the structure of the universal circuit $\hat{\mathcal{U}}$ and compute any other T-size circuit $F(x, K)$. As a result one can easily extract the identifier K and completely learn the function. We resolve this problem via a novel use of *authenticators.*

Using authenticators. Assume that each box has two slots, such that addition and multiplication are performed separately for each slot, and a final box can be opened (zero-tested) if the contents of both slots equal to zero. We keep the second slot as in the previous suggestion. As for the first slot, we generate $n + m$ random authenticators $y_1, \ldots, y_{n+m}$ and assign them to the boxes such that the zero-box and the one-box that correspond to the same input variable x_i share the same authenticator y_i. In addition, let us add another box (labeled by $\mathbf{v}_0$ and referred to as a "nullifier") that contains the pair $(y_0, 0)$ where $y_0 = \hat{\mathcal{U}}(y_1, \ldots, y_{n+m})$, and let us increase the bound

[5]The difference between $\mathcal{URG}$ and $\mathcal{MRG}$ will arise by putting different syntactic restrictions on the class of "legal" polynomials A. Furthermore, an efficient simulator corresponds to VBB security while inefficient simulator corresponds to indistinguishability obfuscation.

on the number of operations to $T+1$. Given an input x, we can apply $\hat{\mathcal{U}}$ to the boxes $\mathbf{v}_{i,x_i}$, obtain a box with the pair $(\hat{\mathcal{U}}(y),\hat{\mathcal{U}}(x,K))$, subtract the result from the last $\mathbf{v}_0$ box and check for zero.

In terms of security, we are now protected from attacks that respect the input x, i.e. use either only the zero-box or only the one-box for each input bit. Specifically, if the adversary applies some $(n+m+1)$-variate polynomial

$$F(V_0,(V_1,\ldots,V_m),(V_1',\ldots,V_n')) \neq \Big(\hat{\mathcal{U}}((V_1',\ldots,V_n'),(V_1,\ldots,V_m)) - V_0\Big)$$

to the boxes $\mathbf{v}_0,(\mathbf{v}_1,\ldots,\mathbf{v}_m),(\mathbf{v}_{1,x_1},\ldots,\mathbf{v}_{n,x_n})$ for some $x \in \{0,1\}^n$, then the result is almost surely non-zero. To see this, let us focus on the first slot of $F(\mathbf{v}_0,(\mathbf{v}_1,\ldots,\mathbf{v}_m),(\mathbf{v}_{1,x_1},\ldots,\mathbf{v}_{n,x_n}))$ and substitute $y_0 = \hat{\mathcal{U}}(y_1,\ldots,y_{n+m})$. Since the adversary is compatible with the GES rules, it follows that the polynomial F simplifies to a non-trivial polynomial over the random values $y_1,\ldots,y_{n+m}$ of degree at most T. Therefore (by Schwartz-Zippel) it vanishes with negligible probability.[6]

From boxes to GES. We adopt the above outline to the GES setting where the two-slot boxes are emulated via the use of a composite-order ring $\mathcal{R} = \mathbb{Z}_{p_1} \times \mathbb{Z}_{p_2}$ where p_1, p_2 are two (large) co-prime integers. The labels of the boxes correspond to GES levels. For now, assume that all levels are multisets over the universe $\{1,\ldots,n+m+1\}$, and thus can be represented by integer-valued vectors in $\mathbb{N}^{n+m+1}$. Further, assume that for $i \in [n]$ we set $\mathbf{v}_{i,0} = \mathbf{v}_{i,1} = \mathbf{e}_i$, for $i \in [m]$ we take $\mathbf{v}_i = \mathbf{e}_{n+i}$ and set $\mathbf{v}_0 = \mathbf{e}_{n+m}$ where $\mathbf{e}_j$ is the j-th unit vector.

Ensuring level compatibility via El-Gamal encoding. Note that there are still several technicalities. First, unlike the simplified boxes model, in the GES setting, addition can be applied only to encodings from the same level. We solve this problem by representing each value $w \in \mathcal{R}$ by a pair $([r]_{\mathbf{v}}, [r \cdot w]_{\mathbf{v}})$ where $r \xleftarrow{R} \mathcal{R}$ is a (unique) randomizer. This "El-Gamal" encoding (which was also used in prior works, e.g. [BR13]) naturally supports addition and multiplication. Namely, if two ring elements w, w' are in El-Gamal form $([r]_{\mathbf{v}}, [rw]_{\mathbf{v}})$ and $([r']_{\mathbf{v}'}, [r'w']_{\mathbf{v}'})$ then their sum can be computed by "cross-multiplication" and their product can be computed by computing component-wise product. As a result, we can evaluate the universal evaluator $\hat{\mathcal{U}}$ on the encoded inputs and subtract at the end, the nullifier which is labeled by $\mathbf{v}_0$. The outcome of such an "honest execution" always reaches the same level, which is defined to be the zero-testing level $\mathbf{v}_{\mathrm{zt}}$.

Coping with input-mixing attacks. The resulting construction is vulnerable to "input-mixing" attacks in which the adversary does not respect any input x, i.e. uses two boxes $\mathbf{v}_{i,0}, \mathbf{v}_{i,1}$ which correspond to the same input variable. We solve this issue via the use of *straddling sets*, following the approach of [BGK+14]. Roughly speaking, straddling sets force consistency by making sure that input-mixing attacks cannot reach the top (zero-testing) level. Details follow.

We "lift" the space of levels from integer-valued vectors of length $(n+m+1)$ to longer vectors whose length is three times larger. In fact, it will be convenient to view the new levels as integer-valued *matrices* of dimension $3 \times (n+m+1)$. (We provide a high level outline below, see also Figure 1 for more complete details.) The levels of the program identifiers, the level of the nullifier and the zero-testing level, are all lifted from vectors to matrices by duplicating the vector three times and placing a copy in each row of the matrix. The "lifting" of the input encodings is a bit

[6]The above argument is somewhat inaccurate as one has to take into account the case where F is a multiple of $(\hat{\mathcal{U}}(\cdots) - V_0)$. A formal proof appears in Section 5.

more subtle. The level $\mathbf{v}_{i,0}$ of the i-th zero-input is "lifted" from $\mathbf{e}_i$ to a matrix whose *first* row is the all-zero vector and its *second and third* rows contain copies of $\mathbf{e}_i$. The level $\mathbf{v}_{i,1}$ of the i-th one-input is modified similarly except that the copies of $\mathbf{e}_i$ are placed in the *first and second* rows and the *third* row is taken to be the zero vector. More explicitly:

$$\mathbf{v}_{i,0} = \begin{bmatrix} 0 \cdots & 0 & \cdots 0 \\ 0 \cdots & 1 & \cdots 0 \\ 0 \cdots & 1 & \cdots 0 \end{bmatrix}, \quad \mathbf{v}_{i,1} = \begin{bmatrix} 0 \cdots & 1 & \cdots 0 \\ 0 \cdots & 1 & \cdots 0 \\ 0 \cdots & 0 & \cdots 0 \end{bmatrix}.$$

As a result, any input-mixing attack that combines $\mathbf{v}_{i,1}$ with $\mathbf{v}_{i,0}$ reaches a level whose i-th column consists of at least two distinct elements, and so such a computation cannot reach the zero-testing level.

Unfortunately, an "honest execution" does not reach the zero level as well. Indeed, an honest execution that corresponds to an input $x = (x_1, \ldots, x_n)$ ends up in a level $\mathbf{v} \in \mathbb{N}^{3\times(n+m+1)}$ whose i-th column has a zero either at the first entry (if $x_i = 0$) or at the third entry (if $x_i = 1$). The other two entries of the column agree with the corresponding entries of the zero-testing level $\mathbf{v}_{\mathrm{zt}}$. We fix this problem by providing additional "dummy" encodings that will be used to push the level $\mathbf{v}$ of an honest execution up to $\mathbf{v}_{\mathrm{zt}}$. For this, we need dummy encodings at levels $\hat{\mathbf{v}}_{i,0}$ and $\hat{\mathbf{v}}_{i,1}$ where the matrix $\hat{\mathbf{v}}_{i,0}$ (resp., $\hat{\mathbf{v}}_{i,1}$) agrees with $\mathbf{v}_{\mathrm{zt}}$ in the first position (resp., third position) of the i-th column and is filled with zeroes in all other locations.

$$\hat{\mathbf{v}}_{i,0} = \begin{bmatrix} 0 \cdots & M[i] & \cdots 0 \\ 0 \cdots & 0 & \cdots 0 \\ 0 \cdots & 0 & \cdots 0 \end{bmatrix}, \quad \hat{\mathbf{v}}_{i,1} = \begin{bmatrix} 0 \cdots & 0 & \cdots 0 \\ 0 \cdots & 0 & \cdots 0 \\ 0 \cdots & M[i] & \cdots 0 \end{bmatrix}.$$

The content of the dummy encodings will be random and, for compatibility, it should be given in El-Gamal encoding. Specifically, for each input $i \in [n]$ we choose a random value $\hat{w}_i$, encode it twice using (fresh) El-Gamal encoding and place one copy at the level $\hat{\mathbf{v}}_{i,0}$ ("the i-th zero dummy input") and another copy at level $\hat{\mathbf{v}}_{i,1}$ ("the i-th one dummy input"). In addition, we update the value of the nullifier by multiplying it with the product $\prod_i \hat{w}_i$ of all dummy values. Now we can run an honest execution over an input $x = (x_1, \ldots, x_n)$, multiply it with the product of all the n dummy inputs that correspond to the selected inputs $(x_1, \ldots, x_n)$, and subtract the nullifier. The result reaches the zero-testing level and so correctness holds. Further, one can show that input-mixing attacks do not lead to $\mathbf{v}_{\mathrm{zt}}$ even at the presence of the additional dummy inputs. This essentially completes the description of our basic obfuscator SimpleObf. (A detailed and formal description appears in Section 4.)

Low-level zero-testing attacks. Interestingly, SimpleObf is completely broken if low-level zero-testing is allowed. Recall that the authenticator y_i is shared among the zero and the one encodings of the i-th input. Therefore, by subtracting the El-Gamal encodings of these elements, we zero-out the authenticator y_i and get a low-level El-Gamal encoding of an element that has zero in the first (authentication) slot and has one in the second slot. We can now multiply the El-Gamal encoding of this element with, say, the El-Gamal encoding of K_1, the first bit of the identifier. This will zero-out the authenticator of K_1, and we will get an encoding of zero if and only if $K_1 = 0$. Therefore, we can recover K_1 (or, more generally, fully recover the identifier K) using low-level zero-testing.

Solving this issue is the main technical challenge addressed by our more robust obfuscator RobustObf. As a first step, we add more slots to the encoding (using $(n + 2)$ subrings $\mathcal{R} =$

$\mathbb{Z}_{p_1} \times \ldots \cdots \mathbb{Z}_{p_{n+2}}$) and make sure that the pair of encodings which share the same y_i, hold distinct (random) values on all other slots. In order to preserve the functionality we must allow the honest evaluator to zero-out the additional slots (while preventing the adversary from doing so in a low-level). To this end, we publish some auxiliary elements (more "dummy" values) $\hat{w}_i$ whose i-th slot is zero. We publish two copies of each $\hat{w}_i$, each in a different level $\hat{\mathbf{v}}_{i,0}, \hat{\mathbf{v}}_{i,1}$, and an appropriate straddling set structure that guarantees that the $\hat{\mathbf{v}}_{i,0}$ copy can only interact with $\mathbf{v}_{i,0}$ and vice versa. Now, if the previous attack is sought, the attacker will attempt to subtract $\mathbf{v}_{i,0}$ from $\mathbf{v}_{i,1}$, but then it will need to multiply by one of $\hat{\mathbf{v}}_{i,0}$ or $\hat{\mathbf{v}}_{i,1}$. Since both operations are forbidden by the straddling sets, the attack seems to be prevented.

Preventing early nullification via shifted El-Gamal encoding. We notice that in order for the previous fix not to interfere with functionality, the new nullifier w_0 (which combines the original nullifier y_0 with the product $\prod_i \hat{w}_i$) must take the value 0 in all of the new slots. The reason is that after the honest evaluator finishes multiplying with the $\hat{w}$ values, it needs to compare against w_0. This leaves us vulnerable to an attacker that will use w_0 instead of the $\hat{w}$ to zero out coordinates ahead of time. To solve this last problem, we present our final trick, the *shifted El-Gamal encoding.* Instead of encoding $([r]_{\mathbf{v}}, [rw]_{\mathbf{v}})$, we will now use $([r]_{\mathbf{v}}, [rw]_{\mathbf{v}+\mathbf{v}^*})$, where $\mathbf{v}^*$ is a special vector used by all of the encodings. The result of this change is that now, if addition/subtraction is performed, the $\mathbf{v}^*$ part of the result is the same as of the operands, but if multiplication is performed, the $\mathbf{v}^*$ part is the sum of the $\mathbf{v}^*$'s of the operands. Therefore the $\mathbf{v}^*$ part keeps track of the multiplicative degree of the evaluation process. Finally, the element w_0 will be encoded as $([r]_{\mathbf{v}_0}, [rw_0]_{\mathbf{v}_0+D\mathbf{v}^*})$, where D is the total multiplicative degree of our evaluation process. This means that one can only add/subtract with w_0, and never multiply (otherwise the $\mathbf{v}^*$ multiple goes beyond D and we set the zero-test level to not allow this). This prevents misuse of w_0 and completes the description of RobustObf.

See Section 4 for a detailed description of the construction and Section 5 for the proof of security.

1.3 Related Works

Comparison with Prior State of the Art. Ananth et al. [AGIS14] (hereafter referred to as AGIS) explored the efficiency of obfuscating formulae. They considered two settings. One where the formula is represented as a sequence of variables and gates, and another more similar to our formulation where there is a universal evaluator (in the form of a formula in their case), and the specific function is specified as a key to this evaluator. In the latter case, which is more relevant for the sake of comparison, they show how to obfuscate classes with formula size L with obfuscated program size and complexity almost as low as $O(L)$. This is in comparison to previous methods that used Barrington's theorem and achieved $O(L^2)$ for balanced formulae or $O(L^{3.64})$ for unbalanced. To compare the complexity of our construction with that of AGIS, we need to consider two aspects. One is the length of the obfuscated program in terms of GES elements, i.e. how many elements are required in order to represent the obfuscation of a certain program. The other is the size of the GES element itself, since we require composite order GES whereas the AGIS construction does not.[7] We note that the latter is harder to compare since our understanding of the security of GES candidates is quite incomplete, and furthermore new GES candidates could possibly emerge

[7] We focus on the *length* of the obfuscated program. One could also consider the *running time* of the obfuscated program, but the two metrics are usually very strongly correlated.

in the future. Keeping this in mind, let us address each of these aspects in the comparison. As a running example, let us consider obfuscating a class of functions whose evaluation function F can be computed by a balanced formula of depth d and size $L = O(2^d)$. Recalling that the input to the evaluator consists of an m-bit index and an n-bit input, we let $N = n + m$ denote the total input length to the evaluator (and keep in mind that $N \leq L$).

- **Number of GES Elements.** The AGIS construction requires roughly $L^{1+\varepsilon}$ elements, proportionally to the size of the formula. Our construction requires $4N$ elements, proportionally to the input length of F. Since L is at least linearly larger than N (and in many interesting cases polynomially larger), our construction is advantageous in this aspect.

- **Size of GES Elements.** The difference here mainly stems from the different number of CRT slots that are required by the different obfuscators. For our obfuscators, SimpleObf requires 2 CRT slots, and RobustObf requires $(n+2)$ slots. The AGIS obfuscator does not require CRT at all, so we may think about it as requiring a single CRT slot. The number of levels scales linearly with 2^d (in our construction) or linearly with the formula-size L (in AGIS). Under the assumption that F is a balanced formula, both quantities are of comparable magnitude.[8]

  The effect of increasing the number of slots on the complexity and security of GES is not entirely clear. We believe that it is conservative to assume that the length of a GES element grows at most linearly with the number of slots, while keeping the level of security roughly unchanged (and assuming the number of levels is the same). Under this assumption, the length of each GES element in our schemes is $O(1)$ greater than in AGIS for SimpleObf, or $O(n)$ greater than AGIS in the case of RobustObf.

In conclusion, we see that SimpleObf could plausibly be asymptotically competitive with AGIS for obfuscating balanced formulas, potentially saving an L/N factor in the size of the obfuscated program. Even RobustObf can be competitive for balanced formulas of super-quadratic size ($L > \Omega(N^2)$). For general unbalanced formulas, the situation is more complicated, and the final verdict (in terms of total bit complexity) depends on the exact relationship between the size, depth, and input length of F, and the exact polynomial dependency of the GES encoding length in the number of levels.

Other Uses of Composite Order GES. In another line of related works, composite order graded encoding schemes have been used by Gentry, Lewko and Waters [GLW14] and by Gentry, Lewko, Sahai and Waters [GLSW14] to introduce improved *security reductions* for witness encryption and for obfuscation (respectively). In particular they showed that in this setting one can construct a witness encryption scheme or an obfuscator, and prove security in the standard model based on exponential hardness assumptions.

Concurrent and Independent Work. In a very recent concurrent and independent work, Zimmerman [Zim14] presented an obfuscator which is almost identical to our simpler obfuscator SimpleObf. Zimmerman also presents applications for this new obfuscation method for circuits, e.g. for functional encryption or multi-bit-output obfuscation. Security is proven in a generic

[8]This means that the *length* of each GES element in either construction scales with 2^d. This is quite undesirable and one could hope that future construction will be able to bring it down to poly(d), but regardless, it is comparable between the two approaches.

model where zero testing below the last level is impossible, similar to our $\mathcal{MRG}$ oracle. Both the obfuscator from [Zim14] and our SimpleObf are completely broken in a more challenging model where it is possible to test for zero at low levels. Our second obfuscator RobustObf addresses this issue and provides security in the more challenging setting represented by the GES oracle $\mathcal{URG}$, at the expense of being less efficient. On the other hand, we only prove that our obfuscators are secure indistinguishability obfuscator in the generic model,[9] whereas [Zim14] proves the more stringent notion of virtual black box security.

Road map. Section 2 defines Graded Encoding over Composite Order Groups and ideal GES oracles. Section 3 defines GES-based obfuscation, suggests two alternative security definitions (standard oracle-based definition and algebraic security) and shows that one implies the other. Section 4 describes our new constructions and Section 5 is devoted to the proof of their security.

Acknowledgments. We thank the reviewers of TCC 2015 and Journal of Cryptology for their insightful and detailed comments. We also thank Mark Zhandry for highlighting a point in our analysis that required further clarification.

2 Graded Encoding over Composite Order Groups

2.1 General Notation

Partial Order of Natural Valued Vectors. For an integer $\tau \in \mathbb{N}$, we view vectors in $\mathbb{N}^\tau$ as multisets over the universe $[\tau]$. Correspondingly, we define a partial ordering "$\leq$" on vectors $\mathbb{N}^\tau$, which has the same semantics as inclusion "$\subseteq$" for the respective multisets. In particular, we say that $\mathbf{v} \leq \mathbf{w}$ if for all $i \in [\tau]$ it holds that $\mathbf{v}[i] \leq \mathbf{w}[i]$. If there exists a coordinate i for which the above does not hold, we say that $\mathbf{v} \not\leq \mathbf{w}$. We note that since our vectors are defined over the naturals, this relation is monotonous: If $\mathbf{v} \leq \mathbf{w}$ then for all $\mathbf{w}' \in \mathbb{N}^\tau$ it also holds that $\mathbf{v} \leq (\mathbf{w}+\mathbf{w}')$, and dually if $\mathbf{v} \not\leq \mathbf{w}$ then for all $\mathbf{v}' \in \mathbb{N}^\tau$ it holds that $(\mathbf{v}+\mathbf{v}') \not\leq \mathbf{w}$.

CRT representation. Let $\sigma \in \mathbb{N}$, let $p_1, \ldots, p_\sigma$ be pairwise coprime numbers and let $P = \prod_{i=1}^{\sigma} p_i$. Considering the ring $\mathbb{Z}_P$, the Chinese Remainder Theorem (CRT) asserts that there is an isomorphism $\mathbb{Z}_P \cong \mathbb{Z}_{p_1} \times \cdots \times \mathbb{Z}_{p_\sigma}$ such that if $a \cong (a_1, \ldots, a_\sigma)$ and $b \cong (b_1, \ldots, b_\sigma)$, then $a + b \cong (a_1 + b_1, \ldots, a_\sigma + b_\sigma)$ and $a \cdot b \cong (a_1 \cdot b_1, \ldots, a_\sigma \cdot b_\sigma)$. For a given isomorphism, we will denote by $a[\![i]\!]$ the component $a_i = a \pmod{p_i}$.

2.2 Syntax

We begin with the definition of a graded encoding scheme in composite order groups. As explained in the introduction, such a scheme maps a pair that consists of a ring element and a multiset, into a collection of strings where each string is viewed as an encoding of the pair. We begin by formalizing the arithmetic operations induced on pair of encoded elements (Definition 2.1), and continue (Definition 2.2) by describing the type of interface provided by such a scheme (encoding an element, performing arithmetic operations on encoding, applying some limited zero-testing, and so on).

[9]We do not see an obstacle for proving VBB, but it seems more technically involved.

The following definitions are adapted from [GGH13a] and follow-up works, but our notation deviates somewhat from that of some previous work.

Definition 2.1 (Graded Encoding Scheme). *Let $\mathcal{R}$ be a ring, and let $\mathbf{v}_{zt} \in \mathbb{N}^\tau$ be an integer vector of dimension $\tau \in \mathbb{N}$. A graded encoding scheme for $\mathcal{R}, \mathbf{v}_{zt}$ is a collection of sets $\{[\alpha]_\mathbf{v} \subset \{0,1\}^* : \mathbf{v} \in \mathbb{N}^\tau, \mathbf{v} \leq \mathbf{v}_{zt}, \alpha \in \mathcal{R}\}$ with the following properties:*

1. *For every index $\mathbf{v} \leq \mathbf{v}_{zt}$, the sets $\{[\alpha]_\mathbf{v} : \alpha \in \mathcal{R}\}$ are disjoint, and so they are a partition of the indexed set $[\mathcal{R}]_\mathbf{v} = \bigcup_{\alpha \in \mathcal{R}} [\alpha]_\mathbf{v}$. We slightly abuse notation and often denote $a = [\alpha]_\mathbf{v}$ instead of $a \in [\alpha]_\mathbf{v}$.*
2. *There are binary operations "+" and "−" such that for all $\mathbf{v} \in \{0,1\}^\tau$, $\alpha_1, \alpha_2 \in \mathcal{R}$ and for all $u_1 = [\alpha_1]_\mathbf{v}$, $u_2 = [\alpha_2]_\mathbf{v}$:*
$$u_1 + u_2 = [\alpha_1 + \alpha_2]_\mathbf{v} \quad \textit{and} \quad u_1 - u_2 = [\alpha_1 - \alpha_2]_\mathbf{v} \ ,$$
where $\alpha_1 + \alpha_2$ and $\alpha_1 - \alpha_2$ are addition and subtraction in $\mathcal{R}$.
3. *There is an associative binary operation "×" such that for all $\mathbf{v}_1, \mathbf{v}_2 \in \mathbb{N}^\tau$ such that $\mathbf{v}_1 + \mathbf{v}_2 \leq \mathbf{v}_{zt}$, for all $\alpha_1, \alpha_2 \in \mathcal{R}$ and for all $u_1 = [\alpha_1]_{\mathbf{v}_1}$, $u_2 = [\alpha_2]_{\mathbf{v}_2}$, it holds that*
$$u_1 \times u_2 = [\alpha_1 \cdot \alpha_2]_{\mathbf{v}_1 + \mathbf{v}_2},$$
where $\alpha_1 \cdot \alpha_2$ is multiplication in $\mathcal{R}$.

The above definition does not touch upon the computational aspects of graded encoding schemes, which are described below. We note that there is a difference between the definition below and the ones that are used in the prime-order setting.

Definition 2.2 (Efficient Procedures for Graded Encoding Scheme). *An efficiently computable collection of graded encoding schemes (GES) is characterized by the following efficient algorithms.*

- *Composite-Order Instance Generation:* $\mathsf{InstGen}(1^\lambda, 1^\sigma, \mathbf{v}_{zt}, 1^{\|\mathbf{v}_{zt}\|_1})$ *outputs the set of parameters params, a description of a Graded Encoding Scheme relative to $\mathbf{v}_{zt}$ and relative to a ring $\mathcal{R}$ such that $\mathcal{R} \cong \mathbb{Z}_{p_1} \times \cdots \times \mathbb{Z}_{p_\sigma}$, where all p_i are pairwise coprime numbers, i.e. $\mathcal{R} \cong \mathbb{Z}_N$ for $N = \prod p_i$.*[10] *We typically refer to elements in $\mathcal{R}$ as tuples over $\mathbb{Z}_{p_1} \times \cdots \times \mathbb{Z}_{p_\sigma}$.*

  In addition, the procedure outputs a subset evparams $\subset$ params that is sufficient for computing addition, multiplication and zero testing, but may be insufficient for sampling, encoding or for randomization.

- *Ring Sampler:* $\mathsf{samp}(params)$ *outputs a "level zero encoding" $A \in [a]_\mathbf{0}$ for a nearly uniform $a \xleftarrow{R} \mathcal{R}$.*

- *Sub-Ring Sampler:* $\mathsf{subsamp}(params, i^*)$*, where $i^* \in [\sigma]$ outputs a "level zero encoding" in a CRT sub-ring of $\mathcal{R}$. Namely, it outputs $A \in [a]_\mathbf{0}$ for an element $a \cong (a_1, \ldots, a_\sigma)$, such that a_{i^*} is statistically indistinguishable from uniform in $\mathbb{Z}_{p_{i^*}}$, and for all $i \neq i^*$ it holds that $a_i = 0$.*[11]

[10] In our security model, we will require that each prime factor of N is chosen from a distribution with roughly $(\Omega(\log \|\mathbf{v}_{zt}\|_1) + \omega(\log \lambda))$ bits of entropy. See Section 5.

[11] Let us emphasize that this procedure requires *params* and not *evparams*. This distinction will be important in order to be able to define a security notion that is plausibly achievable by existing candidates. See Remark 2.5.

- *Encode and Re-Randomize:* $\mathsf{encRand}(params, \mathbf{v}, A)$ *takes as input an index* $\mathbf{v} \leq \mathbf{v}_{zt}$ *and* $A = [a]_{\mathbf{0}}$*, and outputs an encoding* $B = [a]_{\mathbf{v}}$*, where the distribution of* B *is (statistically close to being) only dependent on* a *and not otherwise dependent on* A*.*

- *Addition and Negation:* $\mathsf{add}(evparams, A_1, A_2)$ *takes* $A_1 = [a_1]_{\mathbf{v}}, A_2 = [a_2]_{\mathbf{v}}$*, and outputs* $B = [a_1 + a_2]_{\mathbf{v}}$*. (If the two operands are not in the same indexed set, then* add *returns* $\perp$*). We often use the notation* $u_1 + u_2$ *to denote this operation when evparams is clear from the context. Similarly,* $\mathsf{negate}(evparams, A_1) = [-a_1]_{\mathbf{v}}$*.*

- *Multiplication:* $\mathsf{mult}(evparams, A_1, A_2)$ *takes* $A_1 = [a_1]_{\mathbf{v}_1}, A_2 = [a_2]_{\mathbf{v}_2}$*. If* $\mathbf{v}_1 + \mathbf{v}_2 \leq \mathbf{v}_{zt}$*, then* mult *outputs* $B = [a_1 \cdot a_2]_{\mathbf{v}_1 + \mathbf{v}_2}$*. Otherwise,* mult *outputs* $\perp$*. We often use the notation* $A_1 \times A_2$ *to denote this operation when evparams is clear from the context.*

- *Zero Test:* $\mathsf{isZero}(evparams, A)$ *outputs 1 if* $A = [0]_{\mathbf{v}_{zt}}$*, and 0 otherwise.*

Remark 2.3. *Note that the instance generation procedure takes as input not only* $\mathbf{v}_{zt}$ *but also a unary representation of* $\|\mathbf{v}_{zt}\|_1$*. While one could hope for a definition that does not require the latter, for known GES candidates, the running time of instance generation (and all other procedures) in fact scales with* $\|\mathbf{v}_{zt}\|_1$*. Therefore in order to maintain the convention of polynomial run-time in the input length, we provide* $\|\mathbf{v}_{zt}\|_1$ *in unary as input.*

Noisy encodings. In known candidate constructions, encodings are *noisy* and the noise level increases with addition and multiplication operations, so one has to be careful not to go over a specified noise bound. However, the parameters can be set so as to support $O(\|\mathbf{v}_{\mathrm{zt}}\|_1)$ operations, so long as $\mathsf{InstGen}$ is allowed to run in $\mathrm{poly}(\|\mathbf{v}_{\mathrm{zt}}\|_1)$ time, as our function interface compels. This will be sufficient for our purposes and we therefore ignore noise management throughout this manuscript.

Remark 2.4. *Given params, we can use* $\mathsf{subsamp}$ *together with* $\mathsf{add}, \mathsf{negate}, \mathsf{mult}$ *to efficiently generate level-0 encodings of related elements, so long as each of their CRT components can be expressed as a polynomial size arithmetic circuit (with addition, multiplication, negation and constants* $0, 1$*) applied to a set of uniformly distributed variables. More formally,* $\mathsf{subsamp}$ *allows to generate any set of encodings* $A_1, \ldots, A_\ell$ *(where* $A_j = [a_j]_{\mathbf{0}}$*) with the following structure. For each CRT slot* i*, the values* $a_1[\![i]\!], \ldots, a_\ell[\![i]\!]$ *can all be computed by a multi-output arithmetic circuit as above, whose input is a set of uniform random variables. The sets of input variables for each slot* i *are distinct. (Note that performing arithmetic operations on level-zero encodings does not increase their level.)*

For example, in a 2-composite GES, one can generate $[((x_1 + x_2) \cdot x_3, y_1)]_{\mathbf{0}}$*,* $[(x_3 + x_4, y_2)]_{\mathbf{0}}$*,* $[(x_1 \cdot x_2, y_1 + y_2)]_{\mathbf{0}}$*, for uniform and independent* x, y *variables. However, we cannot generate* $[(y_1, x_1)]_{\mathbf{0}}$ *in addition to the above, since the variables of the first and second slot will not be distinct in this case.*

Combining the above with access to $\mathsf{encRand}$ *allows, given params, to encode the aforementioned elements to arbitrary indices* $\mathbf{v} \leq \mathbf{v}_{zt}$*.*

Remark 2.5. *The security of GES will be formally treated in Section 2.3 via an ideal oracle abstraction. For now, let us just mention some concrete necessary requirements. For our application we require that it is intractable to execute* $\mathsf{subsamp}$ *using only evparams and without access to params. Our application involves an adversary that is given a set of encodings and evparams. If* *the adversary is able to perform sub-ring sampling or to modify the level of an encoded element, then our obfuscator will be insecure.*

Further, in our first construction, the adversary should not be able to apply zero-testing to encodings in level $\mathbf{v} < \mathbf{v}_{zt}$, and should not have a way to distinguish low-level zero encodings from nonzero. This in turn means that we must forbid the adversary to run samp, encRand *as well, since these will allow to "lift" an encoding from level $\mathbf{v}$ to level $\mathbf{v}_{zt}$ and run* isZero*. While this may seem like a severe limitation, known candidates appear to have this property.*

Concrete instantiations. The candidate constructions of [GGH13a, CLT13] do not support the above functionality out of the box. Specifically, [GGH13a] only allows $\mathcal{R}$ of prime order, whereas [CLT13] does natively support composite order groups, but its security features are unclear if sub-ring sampling is allowed. This issue has been extensively addressed in [GLW14, Appendix B of full version]. In particular the authors there present a variant of [CLT13] that appears to overcome the aforementioned security issues. This variant supports a σ-product ring $\mathcal{R} \cong (\mathbb{Z}_{p_1} \times \cdots \times \mathbb{Z}_{p_\sigma})$ where the p_i's are composite numbers with large prime divisors. Note that this is compatible with our requirements which allow the p_i's to be non-primes. Furthermore, this variant adheres to the constraints we need to impose as per Remark 2.5. Overall, even though we prove security in the generic model, we do not know of attacks when our obfuscator is instantiated with this candidate.

2.3 Ideal GES oracles

We would like to prove the security of our construction against *generic adversaries*. To this end, we will use the *generic graded encoding scheme* model, adapted from [BR13, BR14a, BR14b, BGK+14], which is analogous to the *generic group model* (see Shoup [Sho97] and Maurer [Mau05]). Intuitively, we would like to guarantee that the encoding of a ring element reveals no information on the element itself (similarly to random oracle evaluation), and so the adversary can manipulate elements only via the GES procedures. One way to formulate this restriction is to prove security relative to an oracle that implements a truly random GES. We focus on two particular (inefficient) GES oracles: the unique random generic encoding scheme oracle $\mathcal{URG}$ and the multiple-encoding random GES oracle $\mathcal{MRG}$. Both variants will be defined with respect to some probability distribution ensemble $\{\mathcal{R}_{\lambda,\sigma,\mathbf{v}_{\mathrm{zt}}}\}$ over rings (with certain entropic properties as mentioned in Footnote 10). The adversary is not allowed to know the identity of the specific ring that was sampled, except through interaction with the GES oracle interface.

The $\mathcal{URG}$ Oracle. Upon initialization of $\mathsf{InstGen}(1^\lambda, 1^\sigma, \mathbf{v}_{\mathrm{zt}}, 1^{\|\mathbf{v}_{\mathrm{zt}}\|_1})$, the oracle $\mathcal{URG}$ samples a ring $\mathcal{R} \overset{R}{\leftarrow} \mathcal{R}_{\lambda,\sigma,\mathbf{v}_{\mathrm{zt}}}$ and encodes each element $a \in \mathcal{R}$ in level $\mathbf{v} \leq \mathbf{v}_{\mathrm{zt}}$ by a pair $(\mathbf{v}, \rho)$, where ρ is selected uniformly and independently among all the strings of length $t = \lambda \log |\mathcal{R}|$. (We emphasize that for every $(a, \mathbf{v})$ a fresh ρ is being selected). The oracle also releases random private/public parameters $params, evparams \in \{0,1\}^*$ which are associated with this encoding. From now on, the oracle supports all the GES-operations with respect to the above encoding. It is not hard to see that the only way that $\mathcal{A}$ can obtain valid encodings is by calls to the oracle $\mathcal{URG}$ (except with negligible probability).

The oracle $\mathcal{URG}$ is practically identical to the random GES oracle of [BR14b], and similarly to that work we will also consider an online variant of $\mathcal{URG}$, or rather a variant that approximates $\mathcal{URG}$ to within negligible statistical distance. This is done by an *online (stateful) polynomial time* *process*, which samples the representations on-the-fly. Specifically, the oracle will maintain a table of entries of the form $(\mathbf{v}, a, \mathsf{label}_{\mathbf{v},a})$, where $\mathsf{label}_{\mathbf{v},a} \in \{0,1\}^t$ is the representation of $[a]_{\mathbf{v}}$ in $\mathcal{URG}$. The table is initially empty. Every time $\mathcal{URG}$ is called for some functionality, it checks that its operands indeed correspond to an entry in the table, in which case it can retrieve the appropriate $(\mathbf{v}, a)$ to perform the operation. If the operands are not in the table, $\mathcal{URG}$ returns $\perp$. Whenever $\mathcal{URG}$ needs to return a value $[a]_{\mathbf{v}}$, it checks whether $(\mathbf{v}, a)$ is already in the table, and if so returns the appropriate $\mathsf{label}_{\mathbf{v},a}$. Otherwise it samples a new uniform label, and inserts a new entry into the table.

When interacting with an adversary that only makes a polynomial number of calls, the online version of $\mathcal{URG}$ is within negligible statistical distance of the offline version (in fact, the statistical distance is exponentially small in λ). This is because the only case when the online oracle implementation differs from the offline one is when when the adversary guesses a valid label that it has not seen (in the offline setting). This can only occur with exponentially small probability due to the sparsity of the labels. The running time of the online oracle is polynomial in the number of oracle calls.

Defining Multiple-encoding random GES. We would like to define a similar random oracle which assigns exponentially many possible encodings for each element in each level. The interface to this oracle has to be defined carefully. Consider, for example, the case where we have three labels A, B, C where $A = [a]_{\mathbf{v}}$,$B = [b]_{\mathbf{v}}$, $C = [c]_{\mathbf{v}}$ and we compute the term $(A + B) \times C$ and the term $A \times C + B \times C$. We have to specify whether the resulting label will be identical or not. We choose the more conservative approach and assume that in such a case the label will be indeed identical. In contrast, the labels of $A+B$ and $A'+B$ should disagree when A, A' are two *independent* labels of a (e.g., both A and A' were generated using two different calls to $\mathsf{encRand}$ on some label $A_0 = [a]_{\mathbf{0}}$). To formalize these requirements we define an online version of the Multiple-encoding Random GES ($\mathcal{MRG}$) oracle.

The (online) $\mathcal{MRG}$ Oracle. The oracle $\mathcal{MRG}$ is initialized similarly to the $\mathcal{URG}$ oracle, except that each ring element $a \in \mathcal{R}$ in level $\mathbf{v} \leq \mathbf{v}_{\text{zt}}$ is encoded by multiple strings of the form $(\mathbf{v}, \rho_i)$ where ρ_i is random string of length $t = (\log |\mathcal{R}| \cdot \lambda^2)$. Whenever a sampling query is made, $\mathcal{MRG}$ samples an element a from $\mathcal{R}$ (or from the appropriate sub-ring), a uniform length-t label, and also generates a new formal variable X_i. It then stores the tuple $(\mathbf{0}, a, X_i, \mathsf{label}_{\mathbf{0},a,X_i})$ in its table. Whenever an $\mathsf{encRand}$ query is made, again a random label and a new formal variable $X_{i'}$ are chosen, and the tuple $(\mathbf{v}, a, X_{i'}, \mathsf{label}_{\mathbf{v},a,X_{i'}})$ is stored. Whenever an "arithmetic" query is made, $\mathcal{MRG}$ looks up the input labels and finds the appropriate labels in its table, and adds or multiplies the respective formal variables (which will now become formal polynomial). Thus, the table will now contain tuples of the form $(\mathbf{v}, a, \mathcal{P}_C(\vec{X}), \mathsf{label}_{\mathbf{v},a,\text{poly}(\vec{X})})$, where $\mathcal{P}_C$ is a formal polynomial represented by a polynomial size arithmetic circuit C, and labels will be the same if there is equivalence between respective polynomials ($\mathcal{P}_{C_1} \equiv \mathcal{P}_{C_2}$, note that this is an efficiently testable condition[12]), and distinct otherwise. Finally, for zero-test queries, $\mathcal{MRG}$ will test whether the actual value is the zero value in $\mathcal{R}$ and respond accordingly.

Both oracles support the standard GES operations with respect to the resulting encodings. We note that $\mathcal{URG}$ (which essentially corresponds to the traditional notion of multilinear maps over

[12]Polynomial Identity Testing of arithmetic circuit can be done in randomized polynomial time (RP) using Schwartz-Zippel over a sufficiently large field.

generic groups) is more robust than $\mathcal{MRG}$ as it gives more power to the adversary (for example it can easily detect if it has two encodings of the same element). Specifically, it is not hard to show that if a construction is secure with respect to $\mathcal{URG}$ then it is also secure with respect to $\mathcal{MRG}$. (Formally, the $\mathcal{MRG}$ oracle can be efficiently emulated using a $\mathcal{URG}$ oracle.)

3 GES-based Obfuscators

In this section we define the notion of GES-based obfuscators. Our definitions somewhat deviate from the more traditional definitions formulated in [BGI+12]. Specifically, to allow a more fine-grained notions of efficiency, we distinguish between the description-length and the time complexity of the obfuscated program. Furthermore, we adopt the definition to the GES setting and distinguish between correctness, which should hold for any syntactically valid (possibly trivial) GES, and security, which should hold with respect to some "ideal" GES oracle. Finally, we show (Section 3.2) that for natural GES-based obfuscators, security with respect to ideal oracles boils down to certain algebraic properties of the obfuscator's output (referred to as *algebraic security*). This abstraction (which was implicit in previous works) allows us to reduce the security of the obfuscator to an algebraic statement that does not directly relate to the interface of the GES.

3.1 Main Definitions

We begin by recalling the notion of efficient function families.

Function family. Let $\mathcal{C} = \{\mathcal{C}_K\}_{K\in\{0,1\}^*}$ be a family of efficiently computable functions. We will assume that $\mathcal{C}$ is represented by a uniform family of polynomial-size *universal evaluation* circuits $\mathcal{U} = \{\mathcal{U}_n\}_{n\in\mathbb{N}}$, where $\mathcal{U}_n$ maps an identifier $K \in \{0,1\}^{m(n)}$ and input $x \in \{0,1\}^n$ to the output $C_K(x)$ (the efficiency of $\mathcal{U}_n$ imposes a polynomial bound on $m(\cdot)$). The *computational complexity* of $\mathcal{C}$ (with respect to the representation $\mathcal{U}$) is the circuit size of $\mathcal{U}$ and the *representation size* of $\mathcal{C}$ is $m(n)$. We say that $\mathcal{C}$ is in $\mathcal{NC}^1$ if $\mathcal{U}$ is a family of polynomial size circuits of logarithmic depth.[13]

GES-based Obfuscators: Syntax. A GES-based obfuscation scheme for a family of efficiently computable functions $\mathcal{C}$ consists of a pair of PPT algorithms: an obfuscator Obf and an evaluator Eval, which have oracle access to a GES. The input to the obfuscator is an identifier $K \in \{0,1\}^{m(n)}$ of a function $C_K \in \mathcal{C}$, a unary representation of the security parameter 1^λ, and an unary representation 1^n of the input length of C_K. The obfuscator outputs an obfuscated program $\hat{C} \in \{0,1\}^*$. The evaluation algorithm Eval maps an obfuscated program $\hat{C}$, an input $x \in \{0,1\}^n$, and an unary representations of the security parameter 1^λ to a string y. We note that the efficiency requirement on the obfuscator Obf implicitly puts a polynomial restriction on the size of the obfuscated program $\hat{C}$.

Correctness should hold with respect to an arbitrary concrete GES instantiation.

Definition 3.1 (Preserving Functionality)**.** *A GES-based obfuscation scheme* (Obf, Eval) *for $\mathcal{C}$ is* functionality preserving *if for every instantiation of GES $\mathcal{G}$, every $n \in \mathbb{N}$, every $C_K \in \mathcal{C}$ where*

[13]We note that the family of all depth-d size-s circuits for some $s(n) \in \text{poly}(n)$ and $d(n) \in O(\log n)$ admit a universal evaluation circuit in $\mathcal{NC}^1$ of size $s(n) \cdot 2^{d(n)}$.

$K \in \{0,1\}^{m(n)}$*, and every* $x \in \{0,1\}^n$*, with all but* $\mathrm{negl}(\lambda)$ *probability over the coins of* $\mathsf{Obf}, \mathsf{Eval}$ *and the GES oracle* $\mathcal{G}$ *it holds that:*

$$\mathsf{Eval}^{\mathcal{G}}(1^\lambda, \hat{C}, x) = C_K(x), \qquad \text{where } \hat{C} \xleftarrow{R} \mathsf{Obf}^{\mathcal{G}}(1^n, 1^\lambda, K).$$

We define Indistinguishability Obfuscator with respect to some (possibly inefficient) GES instantiation. Our definition is formulated in terms of unbounded simulation which is equivalent to the more standard indistinguishability-based definition (cf. [BR14b]).

Definition 3.2 (Indistinguishability Security [BGI+12])**.** *A GES-based obfuscation scheme* $(\mathsf{Obf}, \mathsf{Eval})$ *for* $\mathcal{C}$ *is called an* Indistinguishability Obfuscator *(iO) with respect to some GES instantiation* $\mathcal{G}$ *if for every (non-uniform) polynomial size adversary* $\mathcal{A}$ *and polynomial* p*, there exists a (computationally unbounded) simulator* $\mathcal{S}$*, such that for every* $n \le p(\lambda)$ *and* for every $C_K \in \mathcal{C}$ *where* $K \in \{0,1\}^{m(n)}$*:*

$$\left| \Pr[\mathcal{A}^{\mathcal{G}}(1^\lambda, \hat{C}) = 1] - \Pr[\mathcal{S}^{C_K}(1^{|K|}, 1^n, 1^\lambda) = 1] \right| = \mathrm{negl}(\lambda),$$

where $\hat{C} \xleftarrow{R} \mathsf{Obf}^{\mathcal{G}}(1^n, 1^\lambda, K)$*. If the simulator can be implemented by (non-uniform) polynomial size circuits than the obfuscator is* Virtually Black-Box *(VBB) secure.*

We will instantiate the above definition with the ideal oracles $\mathcal{URG}$ and $\mathcal{MRG}$ defined in Section 2.3.

3.2 Algebraic Security

In this section we present a notion of security that will be easier to work with, and prove its equivalence to security in the random GES model above. This model and the equivalence are implicit in previous works.

Definition 3.3 (Obfuscator in Canonical form)**.** *A GES-based obfuscator is in* canonical form *if it can be presented as follows. (Recall that the obfuscator is given a security parameter* 1^λ*, an input length* 1^n*, and a program identifier* $K \in \{0,1\}^{m(n)}$*.)*

1. *Based on* n*, the obfuscator deterministically generates* $\ell = \ell(n)$ *integer-valued vectors* $\mathbf{v}_1, \ldots, \mathbf{v}_\ell$*, a zero-testing vector* $\mathbf{v}_{zt}$ *s.t.* $\mathbf{v}_i \le \mathbf{v}_{zt}$ *for all* i*, and a ring arity* $\sigma \in \mathbb{N}$*.*

2. *Based on* λ, K, n*, the obfuscator defines, for any ring* $\mathcal{R} \cong \mathcal{R}_1 \times \cdots \times \mathcal{R}_\sigma$*, a joint distribution* $\mathcal{D}^{\mathcal{R}}_\lambda(n, K)$ *over* ℓ *ring elements* $(a_1, \ldots, a_\ell) \in \mathcal{R}^\ell$*. This distribution is defined by a circuit* $\mathcal{D}_\lambda(n, K)$ *(which is not ring-specific) which uses the following gates: addition, subtraction, multiplication, the constant* 1*, and special sampling gates* $\{S_i\}_{i \in [\sigma]}$*. Given a ring* $\mathcal{R}$*, the distribution* $\mathcal{D}^{\mathcal{R}}_\lambda(n, K)$ *is defined by running* $\mathcal{D}_\lambda(n, K)$ *and replacing* S_i *with random elements from the subring* $\mathcal{R}_i$*. We let* $\deg(\mathcal{D}_\lambda(n, K))$ *denote the degree of the formal polynomial defined by the arithmetic circuit* $\mathcal{D}_\lambda(n, K)$ *(with inputs being the sampling gates).*

3. *Then, the obfuscator initializes the GES by running* $(params, evparams) = \mathsf{InstGen}(1^\lambda, 1^\sigma, \mathbf{v}_{zt}, 1^{\|\mathbf{v}_{zt}\|_1})$*, let* $\mathcal{R}$ *be the underlying ring. The obfuscator computes* $([a_1]_{\mathbf{0}}, \ldots, [a_\ell]_{\mathbf{0}})$ *where* $(a_1, \ldots, a_\ell)$ *are distributed according to* $\mathcal{D}^{\mathcal{R}}_\lambda(n, K)$*, as specified in Remark 2.4. It then uses* $\mathsf{encRand}(params, \mathbf{v}_i, [a_i]_{\mathbf{0}})$ *to compute and output the vector of encodings* $([a_1]_{\mathbf{v}_1}, \ldots, [a_\ell]_{\mathbf{v}_\ell})$*, together with the evaluation parameters evparams.*

Overall, such a canonical obfuscator can be defined by the length function $\ell = \ell(n)$, the ring arity $\sigma(n)$, the vectors $V_n = (\mathbf{v}_1, \dots, \mathbf{v}_\ell, \mathbf{v}_{zt})$ and the circuit $\mathcal{D}_\lambda(n, K)$.

Our notion of security will be defined w.r.t an obfuscator in canonical form $(\ell, \sigma, V, \mathcal{D}_\lambda(n, K))$ and a distribution ensemble $\mathcal{R}_{\lambda,\sigma,\mathbf{v}_{\text{zt}}}$ over rings (supposedly induced by a GES construction). Intuitively, an adversary who gets an obfuscated program $([a_1]_{\mathbf{v}_1}, \dots, [a_\ell]_{\mathbf{v}_\ell})$ can choose some polynomial and check if it is evaluated to zero on the ring elements $(a_1, \dots, a_\ell)$. Security should guarantee that such an attack gives no information on the program K beyond what follows from an oracle access to C_K. That is, we would like to have a simulator that given an oracle access to C_K can tell whether a given adversary A evaluates to zero on $\mathcal{D}^{\mathcal{R}}_\lambda(n, K)$, where $\mathcal{R} \overset{R}{\leftarrow} \mathcal{R}_{\lambda,\sigma,\mathbf{v}_{\text{zt}}}$.

We will formalize this notion of security in Definition 3.6, but before that we define a family of ring-independent adversaries. We will focus on the class of purely arithmetic circuits with arbitrary fan-out. These circuits will not have any constants and will contain only input, addition and multiplication gates. Since it contains no constants, it is not ring-specific and one can consider the evaluation of the same circuit over various rings. Formally, each such circuit naturally defines a polynomial with integer coefficients.

Definition 3.4. *A* purely arithmetic circuit *A is a circuit which contains input gates (no fan-in, fan out > 0), an output gate (fan-in 1, fan out 0) and operator gates for addition $(+)$, subtraction $(-)$ and multiplication $(\times)$ with fan-in 2 and fanout > 0. The* size *of A is the number of gates in A. Given a purely arithmetic circuit A with ℓ input gates and a ring $\mathcal{R}$, we let $\mathcal{P}^{\mathcal{R}}_A \in \mathcal{R}[X_1, \dots, X_\ell]$ denote the ℓ-variate polynomial defined by the circuit A by associating a formal variable X_i with each input gate. When the subscript $\mathcal{R}$ is omitted we view $\mathcal{P}_A$ as a polynomial over the integers.*

We will consider adversaries A that respect the GES-indexing, namely, addition and multiplication can be applied only according to the algebra induced by the GES indexing.

Definition 3.5 (V-compatible circuits)**.** *A purely arithmetic circuit A is evaluated over the integer-valued vectors $(\mathbf{v}_1, \dots, \mathbf{v}_\ell)$ via the following recursive process. The i-th input gate takes the value $\mathbf{v}_i$, a multiplication gate with inputs $\mathbf{v}, \mathbf{v}'$ takes the value $\mathbf{v} + \mathbf{v}'$, and an addition (or subtraction) gate with identical inputs $\mathbf{v} = \mathbf{v}'$ takes the value $\mathbf{v}$. If there exists an addition (subtraction) gate with non-identical inputs $\mathbf{v} \neq \mathbf{v}'$ then the circuit is defined to be* syntactically-illegal. *We say that A is* compatible *with $V = ((\mathbf{v}_1, \dots, \mathbf{v}_\ell), \mathbf{v}_{zt})$ if the computation $A(\mathbf{v}_1, \dots, \mathbf{v}_\ell)$ is syntactically legal and the level $\mathbf{v}$ of the output gate is lower or equal to the zero-test level $\mathbf{v}_{zt}$, i.e., $\mathbf{v} \le \mathbf{v}_{zt}$. When $\mathbf{v} = \mathbf{v}_{zt}$ we say that A is* strictly compatible *with V.*

We can now define a simulation-based definition of security which is parameterized by some family of purely arithmetic circuits $\mathcal{A}_\lambda$.

Definition 3.6 (Algebraic Security)**.** *Let $\mathcal{A} = \{\mathcal{A}_{\lambda,n}\}$ be some class of purely arithmetic circuits where every circuit $A \in \mathcal{A}_{\lambda,n}$ has $\ell(n)$ inputs. We say that a canonical obfuscator $(\ell, \sigma, V, \mathcal{D})$, equipped with a ring distribution $\mathcal{R}_{\lambda,\sigma,\mathbf{v}_{zt}}$, is secure against $\mathcal{A}$ if there exists a (possibly unbounded) randomized algorithm $\mathcal{S}$ (simulator) such that for every input length n, function identifier $K \in \{0,1\}^{m(n)}$, and adversary $A \in \mathcal{A}_{\lambda,n}$ we have*

$$\left| \Pr_{\mathcal{R} \overset{R}{\leftarrow} \mathcal{R}_{\lambda,\sigma,\mathbf{v}_{zt}}} [\mathcal{P}^{\mathcal{R}}_A(\mathcal{D}^{\mathcal{R}}_\lambda(n, K)) = 0] - \Pr[\mathcal{S}^{C_K}(1^\lambda, 1^n, A) = 0] \right| \le \text{negl}(\lambda),$$

where the probability in the minuend is taken over a random choice of a ring $\mathcal{R} \xleftarrow{R} \mathcal{R}_{\lambda,\sigma,\mathbf{v}_{zt}}$ and over a random sample from the distribution $\mathcal{D}_{\lambda}^{\mathcal{R}}(n, K)$. By default, we consider security against all classes of $\mathrm{poly}(\lambda)$*-size purely arithmetic circuits $\mathcal{A} = \{\mathcal{A}_{\lambda,n}\}$ (this implicitly imposes a polynomial bound on n as a function of λ) which are strictly compatible with V_n and refer to this notion as* strict algebraic security*. If the above further holds for circuits which are compatible with V_n in a non-strict sense, we say that the obfuscator achieves* algebraic security*.*

We note that the case of efficient simulator $\mathcal{S}$ corresponds to VBB security and the inefficient case to the notion of iO. Also, different choices of adversaries $\mathcal{A}$ may be considered in order to capture the operations accessible for the adversary in other generic models. A larger class may provide stronger security. Note that the class of V_n-compatible adversaries is larger than the class of *strictly* V_n-compatible, and so security against the former ("algebraic security") implies security against the latter ("strict algebraic security").

The following lemma, which is implicit in previous works (cf. [BR14b]), shows that security in the algebraic model implies security in the generic model.

Lemma 3.7. *Let $\mathcal{O} = (\ell, \sigma, V, \mathcal{D})$ be a canonical GES-based obfuscator, and let $\mathcal{R}_{\lambda,\sigma,\mathbf{v}_{zt}}$ be a distribution supported over rings $\mathcal{R} \cong \mathcal{R}_1 \times \cdots \times \mathcal{R}_\sigma$ s.t. for all $i \in [\sigma]$ and for every fixed nonzero multivariate polynomial p of degree at most $\|\mathbf{v}_{zt}\|_1 \cdot \deg(\mathcal{D}_\lambda(n, K))$ it holds that when sampling a ring $\mathcal{R} \xleftarrow{R} \mathcal{R}_{\lambda,\sigma,\mathbf{v}_{zt}}$ and then a random set of variables from the ith subring $\vec{r} \xleftarrow{R} \mathcal{R}_i$, then* $\Pr_{\mathcal{R},\vec{r}}[p(\vec{r}) = 0] = \mathrm{negl}(\lambda)$.

If $\mathcal{O}$ is algebraically secure with respect to $\mathcal{R}_{\lambda,\sigma,\mathbf{v}_{zt}}$ (resp., strict algebraically secure) then it is a secure indistinguishably obfuscator relative to the GES oracle $\mathcal{URG}$ (resp., $\mathcal{MRG}$) over the ring distribution $\mathcal{R}_{\lambda,\sigma,\mathbf{v}_{zt}}$. Furthermore, if the above holds with efficient simulation then the conclusion is strengthened to VBB security in the corresponding model.

Proof. We first show that algebraic security implies security relative to the $\mathcal{URG}$ oracle, and then proceed to the case of strict algebraic security. Our goal is to simulate the view of an adversary $B^{\mathcal{URG}}(1^\lambda, \hat{C})$, where $\hat{C}$ is produced by running $\mathsf{Obf}^{\mathcal{URG}}(1^n, 1^\lambda, K)$, using only oracle access to the function C_K. Recall that $\hat{C}$ consists of the public parameters *evparams* together with the labels $([a_1]_{\mathbf{v}_1}, \ldots, [a_\ell]_{\mathbf{v}_\ell})$ where $\vec{a} = (a_1, \ldots, a_\ell) \xleftarrow{R} \mathcal{D}_\lambda(n, K)$. Since the simulator does not have an access to K, it cannot sample a state $\vec{a}$ from $\mathcal{D}_\lambda(n, K)$ by himself. Instead, we pretend that such a *hidden state* $\vec{a}$ was already sampled and show how to "program" the (online-version of the) $\mathcal{URG}$ oracle consistently with $\vec{a}$, without knowing $\vec{a}$ itself. This emulation (which follows the methodologies of [BR13, BR14a, BR14b, BGK+14]) will employ the algebraic simulator $\mathcal{S}_{\mathrm{alg}}^{C_K}$ promised by Definition 3.6. Details follow.

The simulator. On an input $(1^{|K|}, 1^n, 1^\lambda)$ (and oracle access to C_K), our simulator $\mathcal{S}$ will maintain a list of triples $(A, \mathbf{v}, \rho)$ where A is a purely arithmetic circuit (compatible with V_n) over $\ell = \ell(n)$ formal variables $(\alpha_1, \ldots, \alpha_\ell)$, $\mathbf{v} \in \mathbb{N}^\tau$ is an integer vector and ρ is a string of length $t = (\log |\mathcal{R}_{\lambda,\sigma,\mathbf{v}_{\mathrm{zt}}}| \cdot \lambda)$. Throughout the simulation we will view $(\mathbf{v}, \rho)$ as the level $\mathbf{v}$ encoding that $\mathcal{URG}$ assigns to the ring element obtained by applying A to the hidden state $\vec{a}$.

At the beginning we initialize the list with ℓ triples $(A_i, \mathbf{v}_i, \rho_i)$ where A_i is the formal polynomial which outputs α_i, $\mathbf{v}_i$ is the i-th entry of V_n (the index set used by the obfuscator) and ρ_i is a random string. We generate *evparams* uniformly and initialize the adversary B with the values $\hat{C} = ((\mathbf{v}_i, \rho_i)_{i\in[\ell]}, \mathit{evparams})$. From now on, we assume that all $\mathcal{URG}$ calls of B contain the identifier *evparams*. Next, if B issues an addition query of the form "$(\mathbf{v},\rho)+(\mathbf{v}',\rho')$", we first verify that the labels $(\mathbf{v},\rho)$ and $(\mathbf{v}',\rho')$ both appear in the list and that the operation respects V (i.e., $\mathbf{v}=\mathbf{v}'\leq \mathbf{v}_{\mathrm{zt}}$). If the verification fails we return $\perp$. Otherwise, we traverse the list and find the circuits A and A' which correspond to $(\mathbf{v},\rho)$ and $(\mathbf{v}',\rho')$. For every tuple in the list of the form $(A_0,\mathbf{v},\rho_0)$, we check if the polynomials $(\mathcal{P}_A+\mathcal{P}_{A'})$ and $\mathcal{P}_{A_0}$ agree on the hidden state $(a_1,\ldots,a_{\ell(n)})$. This check is implemented by calling the algebraic simulator $\mathcal{S}_{\mathrm{alg}}^{C_K}$ on the circuit $A^*=(A+A')-A_0$. If the answer is zero we return the value $(\mathbf{v},\rho_0)$; otherwise, if the answer is non-zero for all tuples on the list, we return $(\mathbf{v},\rho)$ where ρ is a new random string, and add to the list a new triple $(A+A',\mathbf{v},\rho)$.

Multiplication and negation queries are performed similarly with the natural modifications (e.g., in the case of multiplication one has to verify that $(\mathbf{v}+\mathbf{v}')\leq\mathbf{v}_{\mathrm{zt}}$, compare against all $(\mathbf{v}+\mathbf{v}')$-level triples, and output/store an encoding at level $\mathbf{v}+\mathbf{v}'$). The case of a zero-test query $(\mathbf{v}_{\mathrm{zt}},\rho)$ is handled directly by querying the algebraic simulator $\mathcal{S}_{\mathrm{alg}}^{C_K}$ on the circuit A which appears in the tuple $(\mathbf{v}_{\mathrm{zt}},\rho,A)$ (again, if such a tuple does not appear in the list we answer $\perp$).

Analysis. Notice that if the algebraic simulator is efficient then so is the simulator $\mathcal{S}$. We claim that, in each call of the adversary, the statistical distance between the simulator and the real view of B grows by a negligible amount, and so if the number of queries is polynomially bounded the overall statistical distance is negligible.

To see this, first consider the following equivalent description of the actual execution of $B^{\mathcal{URG}}(1^\lambda,\hat{C})$. At the beginning, the obfuscated circuit $\hat{C}$ is sampled just like as in the real execution. At this point, the ring $\mathcal{R}$ is sampled, and the secret vector $\vec{a}$ is sampled from $\mathcal{D}_\lambda^{\mathcal{R}}(n,K)$. The oracle $\mathcal{URG}$ is modified as follows: Given a query to $\mathcal{URG}$, the oracle acts as the simulator described above (including the same book keeping) except that instead of checking whether an arithmetic circuit A^* evaluates to zero, the oracle checks whether $\mathcal{P}_{A^*}(\vec{a})=0$ over the ring $\mathcal{R}$, we refer to this as *a check*.

Next, consider the following hybrid which is identical to the previous experiment except that each check $\mathcal{P}_{A^*}(\vec{a})=0$ is performed with respect to a freshly chosen vector $\vec{a}\xleftarrow{R}\mathcal{D}_\lambda^{\mathcal{R}}(n,K)$ (but the ring remains the same).

We claim that, except with negligible probability, each such check returns the same outcome as in the previous experiment. Consider the distribution $\mathcal{P}_{A^*}(\mathcal{D}_\lambda^{\mathcal{R}}(n,K))$ and, for $i\in[\sigma]$, let $\mathcal{P}_{A^*}(\mathcal{D}_\lambda^{\mathcal{R}}(n,K))[\![i]\!]$ denote its restriction to the i-th sub-ring. The key observation is that this distribution can be written as the output of a polynomial Q_i applied to a uniform vector of elements in the subring $\mathcal{R}_i$, where the degree of Q_i is at most $\deg(A^*)\cdot\deg(\mathcal{D}_\lambda(n,K))\leq\|\mathbf{v}_{\mathrm{zt}}\|_1\cdot d$. (Indeed, Q_i is simply the composition of A^* with the polynomial obtained by taking the circuit $\mathcal{D}_\lambda(n,K)$ and replacing each S_i sampling gate with an input gate and any other sampling gate S_j with the constant 0.) Observe that if, for every sub-ring, the corresponding polynomial Q_i is identically zero over $\mathcal{R}_i$, then the check $\mathcal{P}_{A^*}(\vec{a})=0$ passes for any $\vec{a}$ in the support of $\mathcal{D}_\lambda^{\mathcal{R}}(n,K)$. Otherwise, consider some Q_i which is not identically zero over $\mathcal{R}_i$. Then

$$\Pr_{\vec{a}\xleftarrow{R}\mathcal{D}_\lambda^{\mathcal{R}}(n,K)}[\mathcal{P}_{A^*}^{\mathcal{R}}(\vec{a})=0]=\mathrm{negl}(\lambda),$$

since in the i-th sub-ring we get an evaluation of Q_i on a random input, which, by the conditions of the lemma, zeros out only with negligible probability.

At this point, each check is computed using a uniform and independent $\vec{a}$ but all with the same ring $\mathcal{R}$. We now further consider a sequence of hybrids where in the j-th hybrid, the first j checks are replaced with calls to $\mathcal{S}_{\text{alg}}^{C_K}$ on the respective A^*. The zeroth hybrid is identical to the above and the final hybrid where all checks are replaced with algebraic simulator calls, is exactly the view generated by our simulator $\mathcal{S}$. We claim that each two adjacent hybrids are indistinguishable by the algebraic security property. To see this, we note that in the j-th hybrid, the $(j+1)$-th check specifies a circuit A^* which is independent of the underlying ring, since the adversary's view up to this point is indeed independent of $\mathcal{R}$. This allows us to plug in this A^* as the algebraic security simulator challenge.

Strict Algebraic Security. Let us move to the case of algebraic security with respect to *strictly* V_n-compatible adversaries. Recall that in this case the algebraic simulator cannot be applied to adversaries whose output is strictly lower than $\mathbf{v}_{\text{zt}}$. This seems problematic as in the course of the simulation we had to check the equivalence of a pair of purely arithmetic circuits A and B whose output level is below $\mathbf{v}_{\text{zt}}$. More precisely, to emulate the $\mathcal{URG}$ oracle we had to check whether A and B are likely to agree on the hidden state $(a_1, \ldots, a_\ell)$. This problem vanishes if we only attempt to emulate the $\mathcal{MRG}$ oracle (and so achieve weaker form of security), since this oracle associates a fresh encoding with every formally different polynomials (see Section 2.3).

Recall that the course of execution of $B^{\mathcal{MRG}}(1^\lambda, \hat{C})$ is as follows. First, a ring $\mathcal{R}$ is sampled and the vector $(a_1, \ldots, a_\ell)$ is generated using $\mathcal{D}_\lambda^{\mathcal{R}}(n, K)$. Then, the execution of the adversary is determined by a sequence of tests of two types. *Comparison tests* that determine whether arithmetic circuits A, A' compute the same polynomial over $\mathcal{R}$, and *zero tests* that determine whether an arithmetic circuit A evaluates to zero over the entries $(a_1, \ldots, a_\ell)$. Let us gradually modify this experiment so as to finally derive a simulator.

Consider a hybrid where the first i tests are handled differently from above. Specifically, instead of using the same $\mathcal{R}$ all along, we sample a new ring $\mathcal{R} \overset{R}{\leftarrow} \mathcal{R}_{\lambda,\sigma,\mathbf{v}_{\text{zt}}}$ for every test and apply the test relative to this fresh ring. In zero tests this also involves generating new values for $\vec{a}$. We claim that two adjacent hybrids are indistinguishable. Observe that after the first i tests, the adversary's view is independent of the ring $\mathcal{R}$ and therefore the input to the $i+1$ test (circuits A, A' or circuit A) is independent of $\mathcal{R}$. We distinguish between two cases.

- Suppose that the i-th test is a *comparison test*, where we wish to test whether $A - A'$ computes the zero polynomial. The difference between the two hybrids is whether we check this equivalence in the same ring $\mathcal{R}$ used in all subsequent tests, or whether we generate a fresh ring for it. We note that the degree of $A - A'$ is at most $\|\mathbf{v}_{\text{zt}}\|_1 \cdot \deg(\mathcal{D}_\lambda(n, K))$. If $A - A'$ is identically zero, then both hybrids trivially agree. Otherwise, we note that the condition in the lemma guarantees that $A - A'$ can only be zero with negligible probability over the choice of the ring from $\mathcal{R}_{\lambda,\sigma,\mathbf{v}_{\text{zt}}}$. Thus with all but negligible probability, the tests in both hybrids will return $A \not\equiv A'$.

- If the i-th test is a *zero test* we first apply the same reasoning, but now over the composed circuit $A(\mathcal{D}_\lambda(n, K))$. Again, with all but negligible probability, this polynomial is equivalent to zero on $\mathcal{R}$ if and only if it is equivalent to zero on a fresh ring, except with negligible probability. Furthermore, if it is not equivalent to zero, evaluating it on a random input (which is equivalent to evaluating A on a properly distributed $\vec{a}$) will be nonzero with all but negligible probability. This again means that evaluating on the prescribed $\vec{a}$ over $\mathcal{R}$ is equivalent, up to negligible probability, to sampling fresh $\mathcal{R}, \vec{a}$. The indistinguishability of the hybrids follows.

At this point we see that the adversary's view is indistinguishable from a hybrid where each test is answered relative to an independent ring and input vector. However, the answers to zero tests still depend on the polynomial $\mathcal{D}_\lambda(n, K)$ which in turn depends on the obfuscated circuit. Our next hybrid is to answer zero tests using the algebraic simulator. Indistinguishability follows by definition of algebraic security.

Our last hybrid is thus completely independent of the obfuscated circuit. However, it still requires zero testing of arithmetic circuits relative to a random ring in $\mathcal{R}_{\lambda,\sigma,\mathbf{v}_{\mathrm{zt}}}$. Such a test can be implemented efficiently (up to negligible error probability) by sampling a random $\vec{r}$ and evaluating the function on it. This results in an efficient and circuit independent process that produces a distribution indistinguishable from the adversary's. We define our simulator to be the process generating this distribution. $\square$

4 Description of the Obfuscator and Correctness

In this section we provide a formal description of our obfuscators and establish their correctness. The reader is referred to Section 1.2 for an informal overview of the construction.

4.1 Setting and Definitions

Let $\mathcal{C} = \{\mathcal{C}_K\}_{K\in\{0,1\}^*}$ be a family of efficiently computable functions with n-bit inputs, representation size $m = m(n)$ and universal evaluator $\mathcal{U}$. Let $\hat{\mathcal{U}}$ be the arithmetized version of $\mathcal{U}$. Namely an arithmetic circuit with $\{+, -, \times\}$ gates and the constant 1, such that for any ring $\mathcal{R}$, if $(x, K) \in \{0,1\}^{n+m} \subseteq \mathcal{R}^{n+m}$, then $\hat{\mathcal{U}}(x, K) = C_K(x)$. We assume w.l.o.g that $K_m = 1$, i.e. the description K contains (say as its last bit) the constant 1. Under this assumption $\hat{\mathcal{U}}$ does not need to explicitly contain constants, only the gates $\{+, -, \times\}$ and the variables x, K. We let $\deg(\hat{\mathcal{U}})$ denote the degree of the polynomial computed by $\hat{\mathcal{U}}$.

Consider an enumeration of the wires of $\hat{\mathcal{U}}$ in topological order, such that the first $n + m$ wires refer to the wires of the inputs (x, K) . For each wire i, we define a vector $\mathbf{s}_i \in \mathbb{Z}^{n+m+1}$ as follows. If $i \le n + m$, then $\mathbf{s}_i = \mathbf{e}_i$ (the ith indicator vector). For a wire i which is the output wire of a gate whose input wires are j_1, j_2, we define $\mathbf{s}_i = \mathbf{s}_{j_1} + \mathbf{s}_{j_2}$. We define the *multiplicity* of input wire i to be $M[i] = \mathbf{s}_{\mathrm{out}}[i]$, where "out" is the output wire of $\hat{\mathcal{U}}$. (Note that we only used the first $(n + m)$ coordinates of the vectors. The last coordinate will be utilized in the actual construction for the purpose of checking the consistency of the computation.)

4.2 The Obfuscator SimpleObf

We first define a sequence of level vectors ($\mathbf{v}$'s) in $\mathbb{Z}^{(n+m+1)\times 3}$. These vectors will be defined as tensors of vectors from $\mathbb{Z}^{n+m+1}$ and vectors from $\mathbb{Z}^3$. We illustrate the level vectors in Figure 1.

For all $i \in [n]$, $b \in \{0, 1\}$, we define $\mathbf{v}_{i,b}$ as $\mathbf{v}_{i,b} = \mathbf{e}_i \otimes [b, 1, 1 - b]$, where $\mathbf{e}_i$ denotes the i-th indicator vector of dimension $n + m + 1$. We further define $\hat{\mathbf{v}}_{i,b} = \mathbf{e}_i \otimes [(1 - b) \cdot M[i], 0, b \cdot M[i]]$. For all $i \in \{n + 1, \dots, n + m\}$ we define $\mathbf{v}_i = \mathbf{e}_i \otimes [1, 1, 1]$ and similarly $\mathbf{v}_0 = \mathbf{e}_{n+m+1} \otimes [1, 1, 1]$. Lastly, we define $\mathbf{v}_{\mathrm{zt}} = (\mathbf{s}_{\mathrm{out}} + \mathbf{e}_{n+m+1}) \otimes [1, 1, 1]$. We note that for all $x \in \{0, 1\}^n$ it holds that $\mathbf{v}_{\mathrm{zt}} = \mathbf{v}_0 + \sum_{i=1}^{n}(M[i] \cdot \mathbf{v}_{i,x_i} + \hat{\mathbf{v}}_{i,x_i}) + \sum_{i=n+1}^{n+m} M[i] \cdot \mathbf{v}_i$.

<u>Obfuscator SimpleObf:</u>

$$\mathbf{v}_{i,0} = \left[\begin{array}{ccc|c} 0\cdots & 0 & \cdots 0 & 0 \\ 0\cdots & 1 & \cdots 0 & 0 \\ 0\cdots & 1 & \cdots 0 & 0 \end{array}\right], \quad \mathbf{v}_{i,1} = \left[\begin{array}{ccc|c} 0\cdots & 1 & \cdots 0 & 0 \\ 0\cdots & 1 & \cdots 0 & 0 \\ 0\cdots & 0 & \cdots 0 & 0 \end{array}\right], \quad \mathbf{v}_{i} = \left[\begin{array}{ccc|c} 0\cdots & 1 & \cdots 0 & 0 \\ 0\cdots & 1 & \cdots 0 & 0 \\ 0\cdots & 1 & \cdots 0 & 0 \end{array}\right]$$

$$\hat{\mathbf{v}}_{i,0} = \left[\begin{array}{ccc|c} 0\cdots & M[i] & \cdots 0 & 0 \\ 0\cdots & 0 & \cdots 0 & 0 \\ 0\cdots & 0 & \cdots 0 & 0 \end{array}\right], \quad \hat{\mathbf{v}}_{i,1} = \left[\begin{array}{ccc|c} 0\cdots & 0 & \cdots 0 & 0 \\ 0\cdots & 0 & \cdots 0 & 0 \\ 0\cdots & M[i] & \cdots 0 & 0 \end{array}\right]$$

$$\mathbf{v}_0 = \left[\begin{array}{ccc|c} 0 & \cdots & 0 & 1 \\ 0 & \cdots & 0 & 1 \\ 0 & \cdots & 0 & 1 \end{array}\right], \quad \mathbf{v}_{\mathrm{zt}} = \left[\begin{array}{ccc|c} M[1] & \cdots & M[n+m] & 1 \\ M[1] & \cdots & M[n+m] & 1 \\ M[1] & \cdots & M[n+m] & 1 \end{array}\right]$$

Figure 1: The level vectors for obfuscator $\mathsf{SimpleObf}$. All vectors are in $\mathbb{Z}^{(n+m+1)\times 3}$.

- **Input:** Circuit identifier $K \in \{0,1\}^m$ where $C_K \in \mathcal{C}$.
- **Output:** Obfuscated program with the same functionality as C_K.
- **Algorithm:**

1. Instantiate a 2-composite graded encoding scheme

$$(params, evparams) = \mathsf{InstGen}(1^\lambda, 1^2, \mathbf{v}_{\mathrm{zt}}, 1^{\|\mathbf{v}_{\mathrm{zt}}\|_1}) \ .$$

The following items will be based on the ability to efficiently generate distributions over CRT coordinates using *params* (as explained in Remark 2.4).

2. For all $i \in [n]$, $b \in \{0,1\}$, compute random encodings $R_{i,b} = [r_{i,b}]_{\mathbf{v}_{i,b}}$ as well as encodings of $Z_{i,b} = [r_{i,b} \cdot w_{i,b}]_{\mathbf{v}_{i,b}}$, where $w_{i,b} = (y_i, b)$ and y_i is uniform. More explicitly, $(r_{i,b} \cdot w_{i,b})[\![1]\!] = r_{i,b}[\![1]\!] \cdot y_i$, namely a product of two random elements in the appropriate subring, and $(r_{i,b} \cdot w_{i,b})[\![2]\!]$ is either 0 if $b = 0$, or $r_{i,b}[\![2]\!]$ if $b = 1$.[14]

3. For all $i \in [n]$, $b \in \{0,1\}$, compute random encodings $\hat{R}_{i,b} = [\hat{r}_{i,b}]_{\hat{\mathbf{v}}_{i,b}}$ as well as encodings of $\hat{Z}_{i,b} = [\hat{r}_{i,b} \cdot \hat{w}_{i,b}]_{\hat{\mathbf{v}}_{i,b}}$, where $\hat{w}_{i,0} = \hat{w}_{i,1} = \hat{w}_i = (\hat{y}_i, \hat{\beta}_i)$, i.e. $\hat{w}_i$ is uniform but is used in both $\hat{Z}_{i,b}$.

4. For all $i \in \{n+1, \ldots, n+m\}$, compute random encodings $R_i = [r_i]_{\mathbf{v}_i}$ as well as encodings of $Z_i = [r_i \cdot w_i]_{\mathbf{v}_i}$, where $w_i = (y_i, K_{i-n})$, where K_i is the ith bit of the circuit description and y_i is uniform.

5. Compute random encoding $R_0 = [r_0]_{\mathbf{v}_0}$ and $Z_0 = [r_0 w_0]_{\mathbf{v}_0}$, where $w_0 = \left(\prod_{i\in[n]} \hat{w}_i\right) \cdot (y_0, 1)$ and $y_0 = \hat{\mathcal{U}}(y_1, \ldots, y_{n+m})$.

6. The obfuscated program will contain the following:

[14]Recall that, by convention, we refer to ring elements via their CRT representation.

- The evaluation parameters *evparams*.
- For all $i \in [n]$, $b \in \{0,1\}$ the elements $R_{i,b}, Z_{i,b}, \hat{R}_{i,b}, \hat{Z}_{i,b}$.
- For all $i \in \{n+1, \dots, n+m\}$ the elements R_i, Z_i.
- The elements R_0, Z_0.

An important feature of our obfuscator (that will be used in the proof) is that all of the information that depends on the circuit C_K resides in the second element of the CRT representation, and the distribution of the first element is completely independent of C_K.

4.3 The Obfuscator RobustObf

For simplicity of presentation, we assume w.l.o.g that $\hat{\mathcal{U}}$ is such that the inputs to every binary gate (addition or multiplication) have the same total degree (as polynomials in the input variables and program description). This is straightforward to achieve since, as we explained above, we fixed one of the last element of the program description to be equal to 1, i.e. $K_m = 1$. Therefore multiplying by the appropraite power of K_m can be used to balance the input degrees.[15]

We define a sequence of level vectors ($\mathbf{v}$'s) in $\mathbb{Z}^{(n+m+1)\times 4}$. These vectors will be defined as tensors of vectors from $\mathbb{Z}^{n+m+1}$ and vectors from $\mathbb{Z}^4$. We illustrate the level vectors in Figure 2.

For all $i \in [n]$, $b \in \{0,1\}$, we define $\mathbf{v}_{i,b}$ as $\mathbf{v}_{i,b} = \mathbf{e}_i \otimes [b, 1, 1-b, 0]$. We further define $\hat{\mathbf{v}}_{i,b} = \mathbf{e}_i \otimes [(1-b) \cdot M[i], 0, b \cdot M[i], 1]$. For all $i \in \{n+1, \dots, n+m\}$ we define $\mathbf{v}_i = \mathbf{e}_i \otimes [1,1,1,0]$. We define $\mathbf{v}_0 = \mathbf{e}_{n+m+1} \otimes [1,1,1,0]$ and $\mathbf{v}^* = \mathbf{e}_{n+m+1} \otimes [0,0,0,1]$. Lastly, we define $\mathbf{v}_{\text{zt}} = (\mathbf{s}_{\text{out}} + \mathbf{e}_{n+m+1}) \otimes [1,1,1,0] + (\sum_{i=1}^{n} \mathbf{e}_i) \otimes [0,0,0,1] + D \cdot \mathbf{v}^*$, where $D = \deg(\hat{\mathcal{U}}) + n$. We note that for all $x \in \{0,1\}^n$ it holds that $\mathbf{v}_{\text{zt}} = \mathbf{v}_0 + \sum_{i=1}^{n}(M[i] \cdot \mathbf{v}_{i,x_i} + \hat{\mathbf{v}}_{i,x_i}) + \sum_{i=n+1}^{n+m} M[i] \cdot \mathbf{v}_i + D \cdot \mathbf{v}^*$.

<u>Obfuscator RobustObf:</u>

- **Input:** Circuit identifier $K \in \{0,1\}^m$ where $C_K \in \mathcal{C}$.
- **Output:** Obfuscated program with the same functionality as C_K.
- **Algorithm:**

1. Instantiate a $(n+2)$-composite graded encoding scheme

$$(params, evparams) = \mathsf{InstGen}(1^\lambda, 1^{n+2}, \mathbf{v}_{\text{zt}}, 1^{\|\mathbf{v}_{\text{zt}}\|_1}) \ .$$

   The following items will be based on the ability to efficiently generate distributions over CRT coordinates using *params* (as explained in Remark 2.4).

2. For all $i \in [n]$, $b \in \{0,1\}$, compute random encodings $R_{i,b} = [r_{i,b}]_{\mathbf{v}_{i,b}}$ as well as encodings of $Z_{i,b} = [r_{i,b} \cdot w_{i,b}]_{\mathbf{v}_{i,b}+\mathbf{v}^*}$, where $w_{i,b} = (y_i, b, \rho_{i,b,1}, \dots, \rho_{i,b,n})$ and $y_i, \rho_{i,b,j}$ are uniform. In particular, $(r_{i,b} \cdot w_{i,b})[\![2]\!]$ is either 0 if $b = 0$ or $r_{i,b}[\![2]\!]$ if $b = 1$.

[15]While this transformation may incur a significant increase in the degree, the depth remains unchanged up to a constant. Therefore, the total degree is at most exponential in the depth of the original Boolean circuit.

$$\mathbf{v}_{i,0} = \left[\begin{array}{ccc|c} 0\cdots & 0 & \cdots 0 & 0 \\ 0\cdots & 1 & \cdots 0 & 0 \\ 0\cdots & 1 & \cdots 0 & 0 \\ \hline 0\cdots & 0 & \cdots 0 & 0 \end{array}\right], \quad \mathbf{v}_{i,1} = \left[\begin{array}{ccc|c} 0\cdots & 1 & \cdots 0 & 0 \\ 0\cdots & 1 & \cdots 0 & 0 \\ 0\cdots & 0 & \cdots 0 & 0 \\ \hline 0\cdots & 0 & \cdots 0 & 0 \end{array}\right], \quad \mathbf{v}_{i} = \left[\begin{array}{ccc|c} 0\cdots & 1 & \cdots 0 & 0 \\ 0\cdots & 1 & \cdots 0 & 0 \\ 0\cdots & 1 & \cdots 0 & 0 \\ \hline 0\cdots & 0 & \cdots 0 & 0 \end{array}\right]$$

$$\hat{\mathbf{v}}_{i,0} = \left[\begin{array}{ccc|c} 0\cdots & M[i] & \cdots 0 & 0 \\ 0\cdots & 0 & \cdots 0 & 0 \\ 0\cdots & 0 & \cdots 0 & 0 \\ \hline 0\cdots & 1 & \cdots 0 & 0 \end{array}\right], \quad \hat{\mathbf{v}}_{i,1} = \left[\begin{array}{ccc|c} 0\cdots & 0 & \cdots 0 & 0 \\ 0\cdots & 0 & \cdots 0 & 0 \\ 0\cdots & M[i] & \cdots 0 & 0 \\ \hline 0\cdots & 1 & \cdots 0 & 0 \end{array}\right]$$

$$\mathbf{v}_{0} = \left[\begin{array}{ccc|c} 0 & \cdots & 0 & 1 \\ 0 & \cdots & 0 & 1 \\ 0 & \cdots & 0 & 1 \\ \hline 0 & \cdots & 0 & 0 \end{array}\right], \quad \mathbf{v}^{*} = \left[\begin{array}{ccc|c} 0 & \cdots & 0 & 0 \\ 0 & \cdots & 0 & 0 \\ 0 & \cdots & 0 & 0 \\ \hline 0 & \cdots & 0 & 1 \end{array}\right]$$

$$\mathbf{v}_{\mathrm{zt}} = \left[\begin{array}{ccc|ccc|c} M[1] & \cdots & M[n] & M[n+1] & \cdots & M[n+m] & 1 \\ M[1] & \cdots & M[n] & M[n+1] & \cdots & M[n+m] & 1 \\ M[1] & \cdots & M[n] & M[n+1] & \cdots & M[n+m] & 1 \\ \hline 1 & \cdots & 1 & 0 & \cdots & 0 & D \end{array}\right]$$

Figure 2: The level vectors for obfuscator RobustObf. All vectors are in $\mathbb{Z}^{(n+m+1)\times 4}$.

3. For all $i \in [n]$, $b \in \{0,1\}$, compute random encodings $\hat{R}_{i,b} = [\hat{r}_{i,b}]_{\hat{\mathbf{v}}_{i,b}}$ as well as encodings of $\hat{Z}_{i,b} = [\hat{r}_{i,b} \cdot \hat{w}_i]_{\hat{\mathbf{v}}_{i,b}+\mathbf{v}^*}$, where $\hat{w}_{i,0} = \hat{w}_{i,1} = \hat{w}_i = (\hat{y}_i, \hat{\beta}_i, \hat{\rho}_{i,1}, \ldots, \hat{\rho}_{i,n})$, where $\hat{y}_i, \hat{\beta}_i, \{\hat{\rho}_{i,j}\}_{j\neq i}$ are all uniform, but $\hat{\rho}_{i,i} = 0$.

4. For all $i \in \{n+1, \ldots, n+m\}$, compute random encodings $R_i = [r_i]_{\mathbf{v}_i}$ as well as encodings of $Z_i = [r_i \cdot w_i]_{\mathbf{v}_i+\mathbf{v}^*}$, where $w_i = (y_i, K_{i-n}, \rho_{i,1}, \ldots, \rho_{i,n})$, where K_i is the ith bit of the circuit description and $y_i, \rho_{i,j}$ are uniform.

5. Compute random encoding $R_0 = [r_0]_{\mathbf{v}_0}$ and $Z_0 = [r_0 w_0]_{\mathbf{v}_0+D\mathbf{v}^*}$, where $w_0 = \left(\prod_{i\in[n]} \hat{w}_i\right) \cdot (y_0, 1, 0, \ldots, 0)$ and $y_0 = \hat{\mathcal{U}}(y_1, \ldots, y_{n+m})$.

6. The obfuscated program will contain the following:
   - The evaluation parameters *evparams*.
   - For all $i \in [n]$, $b \in \{0,1\}$ the elements $R_{i,b}, Z_{i,b}, \hat{R}_{i,b}, \hat{Z}_{i,b}$.
   - For all $i \in \{n+1, \ldots, n+m\}$ the elements R_i, Z_i.
   - The elements R_0, Z_0.

Note that again all of the information that depends on C_K appears in the second component of $\mathcal{R}$, and the distributions in all other components are independent of K.

4.4 Evaluating an Obfuscated Program

We will now describe the evaluator for our obfuscators SimpleObf and RobustObf. Due to their very similar structure, we are able to present a single evaluator that works for both obfuscators. In the context of SimpleObf we will define $\mathbf{v}^* = \mathbf{0}$ and ignore the last n sub-rings that the ring $\mathcal{R}$ would have had in the case of RobustObf.

As can be seen in the description of our obfuscator above, the obfuscated circuit is encoded in the w variables, and each w variable in turn is encoded relative to an r variable. We first show that these pairs of encodings of r and $r \cdot w$ can be manipulated algebraically while keeping the invariant that each value is encoded relative to an r. This is demonstrated by the following procedure.

Procedure PairOp:

- **Input:** GES evaluation parameters *evparams*, pairs of encodings $\big(R_1 = [r_1]_{\mathbf{v}_1}, Z_1 = [r_1 w_1]_{\mathbf{v}_1 + k\mathbf{v}^*}\big)$, $\big(R_2 = [r_2]_{\mathbf{v}_2}, Z_2 = [r_2 w_2]_{\mathbf{v}_2 + k\mathbf{v}^*}\big)$, operation $\mathsf{op} \in \{\times, +, -\}$.
- **Output:** Pair of encodings $\big(R^* = [r_1 r_2]_{\mathbf{v}_1+\mathbf{v}_2}, Z^* = [r_1 r_2 \cdot (w_1 \;\mathsf{op}\; w_2)]_{\mathbf{v}_1+\mathbf{v}_2+tk\cdot\mathbf{v}^*}\big)$, where $t = 1$ for $\mathsf{op} \in \{+,-\}$ and $t = 2$ for $\mathsf{op} \in \{\times\}$. If $(\mathbf{v}_1 + \mathbf{v}_2 + tk \cdot \mathbf{v}^*) > \mathbf{v}_{\mathrm{zt}}$, the procedure outputs $\perp$.
- **Algorithm:**

1. Compute $R^* = R_1 \times R_2$.
2. If $\mathsf{op} = \times$ compute $Z^* = Z_1 \times Z_2$.
3. If $\mathsf{op} = +$ compute $Z^* = Z_1 \times R_2 + R_1 \times Z_2$.
4. If $\mathsf{op} = -$ compute $Z^* = Z_1 \times R_2 - R_1 \times Z_2$.

We note that PairOp can be applied iteratively to evaluate any arithmetic circuit on pairs of encodings provided that the inputs to every gate have the same degree. The multiplicity of $\mathbf{v}^*$ will be exactly the multiplicative degree of the evaluated circuit. We can now describe our evaluator for obfuscated programs.

Procedure Eval:

- **Input:** Obfuscated program as produced by $\mathsf{SimpleObf}(K)$ for some identifier K:

$$\mathcal{O} = \left(evparams, \{R_{i,b}, Z_{i,b}, \hat{R}_{i,b}, \hat{Z}_{i,b}\}_{\substack{i\in[n],\\ b\in\{0,1\}}}, \{R_i, Z_i\}_{i=n+1}^{n+m}, \{R_0, Z_0\} \right),$$

  input $x \in \{0,1\}^n$.

- **Output:** Value $\mathcal{O}(x) \in \{0,1\}$.
- **Algorithm:**

1. We consider the pairs of elements (R_{i,x_i}, Z_{i,x_i}) for $i \in [n]$, and R_i, Z_i for $i = n+1, \ldots, n+m$. We apply the circuit $\hat{\mathcal{U}}$ on these pairs of encodings as described above, to obtain a pair:

$$R_{\mathcal{U}} = [r_{\mathcal{U}}]_{\mathbf{v}_{\mathcal{U}}}, \quad Z_{\mathcal{U}} = [r_{\mathcal{U}} \cdot w_{\mathcal{U}}]_{\mathbf{v}_{\mathcal{U}} + \deg(\hat{\mathcal{U}})\mathbf{v}^*} \ ,$$

where $\mathbf{v}_{\mathcal{U}} = \sum_{i=1}^{n} M[i] \cdot \mathbf{v}_{i,x_i} + \sum_{i=n+1}^{n+m} M[i] \cdot \mathbf{v}_i$ and

$$\begin{aligned} w_{\mathcal{U}} &= \hat{\mathcal{U}}(w_{1,x_1}, \ldots, w_{n,x_n}, w_{n+1}, \ldots, w_{n+m}) \\ &= (\hat{\mathcal{U}}(y_1, \ldots, y_n, y_{n+1}, \ldots, y_{n+m}), \hat{\mathcal{U}}(x, K), —) \\ &= (\hat{\mathcal{U}}(\vec{y}), C_K(x), —) \end{aligned}$$

where the values denoted by "—" will not matter for correctness so we will not explicitly mention them to avoid cluttering (recall that the simpler obfuscator SimpleObf does not need these values at all).

2. We take the product of the pair of elements $(R_{\mathcal{U}}, Z_{\mathcal{U}})$ with the pairs $(\hat{R}_{i,x_i}, \hat{Z}_{i,x_i})$ to obtain

$$\hat{R}_{\mathcal{U}} = [\hat{r}_{\mathcal{U}}]_{\hat{\mathbf{v}}_{\mathcal{U}}}, \quad \hat{Z}_{\mathcal{U}} = [\hat{r}_{\mathcal{U}} \cdot \hat{w}_{\mathcal{U}}]_{\hat{\mathbf{v}}_{\mathcal{U}} + D\mathbf{v}^*} \ ,$$

where $\hat{w}_{\mathcal{U}} = \prod_{i=1}^{n} \hat{w}_i \cdot w_{\mathcal{U}}$, and

$$\hat{\mathbf{v}}_{\mathcal{U}} = \sum_{i=1}^{n} M[i] \cdot \mathbf{v}_{i,x_i} + \sum_{i=n+1}^{n+m} M[i] \cdot \mathbf{v}_i + \sum_{i=1}^{n} \hat{\mathbf{v}}_{i,x_i} = \mathbf{v}_{\mathrm{zt}} - D\mathbf{v}^* - \mathbf{v}_0 \ .$$

3. We subtract the pair $(\hat{R}_{\mathcal{U}}, \hat{Z}_{\mathcal{U}})$ from the pair (R_0, Z_0), to obtain

$$R' = [r']_{\hat{\mathbf{v}}_{\mathcal{U}} + \mathbf{v}_0}, \quad Z' = [r' \cdot w']_{\hat{\mathbf{v}}_{\mathcal{U}} + D\mathbf{v}^* + \mathbf{v}_0} \ ,$$

and we notice that indeed $(\hat{\mathbf{v}}_{\mathcal{U}} + D\mathbf{v}^* + \mathbf{v}_0) = \mathbf{v}_{\mathrm{zt}}$ and

$$w' = w_0 - \left(\prod_{i=1}^{n} \hat{w}_i\right) \cdot (\hat{\mathcal{U}}(\vec{y}), C_K(x), —) = \left(\prod_{i=1}^{n} \hat{w}_i\right) \cdot (\hat{\mathcal{U}}(\vec{y}) - \hat{\mathcal{U}}(\vec{y}), 1 - C_K(x), —) \ .$$

Recalling that $\prod_{i=1}^{n} \hat{w}_i = (\alpha, \beta, 0, \ldots, 0)$, for some values α, β, we have that

$$w' = (0, \beta(1 - C_K(x)), 0, \ldots, 0) \ .$$

4. Finally, zero testing is applied to Z'. If $\mathsf{isZero}(Z') = 1$ then output 1, otherwise output 0.

---

5 Generic Security of Our Construction

In this section we prove that our constructions are secure when instantiated with a properly chosen ring distribution. Towards this end, we define the notion of *admissible* distributions over composite numbers and, by extension, over rings. Intuitively, a probability distribution $\mathcal{N}_k$ is k-admissible if it samples a poly(k)-bit integer N, with the property that for any a-priori fixed prime p, the probability that p divides N is small, specifically at most 2^{-k}.

Definition 5.1. *A probability distribution $\mathcal{N}$ over the positive integers is k-*admissible *if for every prime p it holds that*

$$\Pr_{N \leftarrow \mathcal{N}}[p \text{ divides } N] \leq 2^{-k} .$$

An ensemble of probability distributions $\{\mathcal{N}_k\}$ is admissible *if, for every k, $\mathcal{N}_k$ is k-admissible and is supported over* poly(k)*-bit integers. An ensemble of probability distributions over rings $\{\mathcal{R}_k\}$ is admissible if, for every k, $\mathcal{R}_k \cong \mathbb{Z}_N$ and the random variable N is k-admissible.*

Remark 5.2 (sampling k-admissible distributions)**.** *A natural way to sample a k-admissible distribution with t prime divisors is to sample t primes uniformly from the set of $O(k + \log t)$-bit prime integers and take their product. Using the density of primes, the probability of hitting any fixed prime is at most 2^{-k}.*

The concrete candidate for composite order GES proposed in [GLW14, Appendix B] indeed samples the ring order from an a-priori high entropy distribution. The public parameters of the scheme reveal the order information theoretically, but they still need to remain computationally hard to find. When formulating a generic model, we do not impose computational restrictions and therefore we model this property via information theoretic hiding of the subring orders. We further note that the proof goes through under a more general requirement as explained in Footnote 17.

Given the above definition, we will prove the following theorems.

Theorem 5.3. *The obfuscator* $\mathsf{SimpleObf}$ *is secure relative to the GES oracle $\mathcal{MRG}$ defined over any* $(4 \log \|\mathbf{v}_{zt}\|_1 + \omega(\log \lambda))$-admissible *ring distribution.*

Theorem 5.4. *The obfuscator* $\mathsf{RobustObf}$ *is secure relative to the GES oracle $\mathcal{URG}$ defined over any* $(4 \log \|\mathbf{v}_{zt}\|_1 + \omega(\log \lambda))$-admissible *ring distribution.*

In Section 5.2 we reduce the proof of both theorems to the existence of an algorithm that tests whether some given polynomial (derived from the adversary) vanishes. Sections 5.3 and 5.4 are devoted to constructing this tester. We begin (in Section 5.1) by collecting some useful algebraic facts regarding polynomials and admissible rings. These facts will be used later in the proof and the reader may choose to skip directly to Section 5.2.

5.1 Useful Algebraic Tools

We will use the following corollary of the Schwartz-Zippel lemma [Sch80, Zip79].

Fact 5.5. *Let $t \in \mathbb{N}$, let $p_1 < \cdots < p_t$ be distinct primes and let $p = \prod_{i=1}^{t} p_i$. Let $Q(x_1, \ldots, x_n) \in \mathbb{Z}[x_1, \ldots, x_n]$ be a multivariate polynomial over the integers of total degree d. Suppose that Q is nonzero modulo* some *p_i's, then*

$$\Pr_{r \xleftarrow{R} \mathbb{Z}_p} [Q(r) = 0 \pmod{p}]$$

is at most $\frac{d}{p_1}$.

Proof. Consider the non-empty set $S = \{i : Q(x) \not\equiv 0 \pmod{p_i}\}$. By CRT, we have

$$\Pr_{r \overset{R}{\leftarrow} \mathbb{Z}_p}[Q(r) = 0 \pmod{p}] = \prod_{i \in [t]} \Pr_r[Q(r) = 0 \pmod{p_i}] \le \prod_{i \in S} \frac{d}{p_i} \le \frac{d}{p_1},$$

where the first inequality is due to Schwartz-Zippel, and the second one holds whenever S is non-empty. $\square$

For a univariate polynomial P, defined over a field, it holds that $(x-a)|P(x)$ if and only if $P(a) = 0$. The following lemma generalizes this fact to the case of multivariate polynomials over the integers

Fact 5.6. *Let* $P(x_1, \ldots, x_n) \in \mathbb{Z}[x_1, \ldots, x_n]$ *and let* $A(x_2, \ldots, x_n) \in \mathbb{Z}[x_1, \ldots, x_n]$ *(however* x_1 *does not appear in* A*). Then*

$$\big((x_1 - A(x_2, \ldots, x_n))|P(x_1, \ldots, x_n)\big) \leftrightarrow P(A(x_2, \ldots, x_n), x_2, \ldots, x_n) \equiv 0 \ .$$

Proof. It is a well known fact that the above holds for polynomials over the rationals (by Bézout's little theorem), however we wish to prove that it holds for polynomials over $\mathbb{Z}$ as well. Let $P, A \in \mathbb{Z}[x_1, \ldots, x_n] \subseteq \mathbb{Q}[x_1, \ldots, x_n]$ be as in the lemma statement. We note that composition of integer polynomials is a special case of composition over the rationals, so the polynomial $P(A(x_2, \ldots, x_n), x_2, \ldots, x_n)$ is defined in the same way over $\mathbb{Z}[x_1, \ldots, x_n]$ and over $\mathbb{Q}[x_1, \ldots, x_n]$.

Since $\mathbb{Q}$ is a field, it holds that $P(A(x_2, \ldots, x_n), x_2, \ldots, x_n) \equiv 0$ if and only if

$$(x_1 - A(x_2, \ldots, x_n))|_{\mathbb{Q}} P(x_1, \ldots, x_n) \ ,$$

in other words there exists $B(x_1, \ldots, x_n) \in \mathbb{Q}[x_1, \ldots, x_n]$ such that

$$P(x_1, \ldots, x_n) \equiv (x_1 - A(x_2, \ldots, x_n)) \cdot B(x_1, \ldots, x_n) \ . \tag{1}$$

We will prove next that in fact all of the coefficients of B are integers, namely $B(x_1, \ldots, x_n) \in \mathbb{Z}[x_1, \ldots, x_n]$, which will prove that

$$(x_1 - A(x_2, \ldots, x_n))|_{\mathbb{Z}} P(x_1, \ldots, x_n) \ ,$$

and finish the proof of the lemma.

To see this, let d be the individual degree of x_1 in P, and define $B_i(x_2, \ldots, x_n) \in \mathbb{Z}[x_1, \ldots, x_n]$ such that $B(x_1, \ldots, x_n) = \sum_{i=0}^{d-1} B_i(x_2, \ldots, x_n) \cdot x_1^i$. We define $P_i(x_2, \ldots, x_n)$ to be such that $P(x_1, \ldots, x_n) = \sum_{i=0}^{d} P_i(x_2, \ldots, x_n) \cdot x_1^i$. Using this decomposition, Eq. (1) implies that $P_d = B_{d-1}$ and, for every $i < d-1$, we have that $P_{i+1} = B_i - A \cdot B_{i+1}$. Note that all P_i have integer coefficients by definition. Therefore B_{d-1} is an integer polynomials (as it is equal to P_d). Since B_i can be written as $P_{i+1} + A \cdot B_{i+1}$, it follows (using induction on i from $d-1$ to 0) that B_i is also an integer polynomial. $\square$

Next we present a bound on the size of the coefficients of a polynomial computed by a purely arithmetic circuit of bounded size and bounded degree.

Fact 5.7. *Let C be a purely arithmetic circuit (as per Definition 3.4) of size s and degree d. Then the ℓ_1 norm of the coefficient vector of the polynomial computed by C is at most 2^{sd}.*

Proof. We prove by induction. The base case is a circuit without gates for which the fact holds immediately. If the output gate of C is a multiplication gate, then consider the two circuits representing the input wires to this gate. These circuit have size $\leq (s-1)$ and degrees d_1, d_2 such that $d_1 + d_2 \leq d$. Therefore by induction we get that the norm is at most $2^{(s-1)d_1} \cdot 2^{(s-1)d_2} \leq 2^{sd}$. If the output gate is an addition or subtraction gate, then the input wires have size $s-1$ and degree at most d, in which case the norm is at most $2^{(s-1)d} + 2^{(s-1)d} \leq 2^{sd}$. □

A polynomial is *free* of some variable or monomial if this variable/monomial does not appear in its expansion. A formal definition follows.

Definition 5.8. *Let $P(X_1, \ldots, X_n)$ be a polynomial. We say that P is X_i-free if all monomials that contain X_i take zero value in P's coefficient vector. We extend this notation to monomials and say that P is $(\prod X_i^{d_i})$-free if all monomials that are divisible by $(\prod X_i^{d_i})$ take zero value in P's coefficient vector. For a set of monomials $\{M_1, \ldots, M_k\}$ we say that P is $\{M_1, \ldots, M_k\}$-free if it is M_j-free for all $j = 1, \ldots, k$.*

Property of admissible rings. Recall that an ensemble of probability distributions over rings $\{\mathcal{R}_k\}$ is admissible if, for all k, $\mathcal{R}_k \cong \mathbb{Z}_N$ and the random variable N is sampled from a k-admissible integer distribution $\mathcal{N}_k$. That is, $\mathcal{N}_k$ is a distribution over poly(k)-bit integers with the property that the min-entropy of every prime factor of $\mathcal{N}_k$ is at least $\Omega(k)$. It is not hard to see that every predefined poly(k)-bit integer x is extremely likely to be co-prime to $y \xleftarrow{R} \mathcal{N}_k$.

Claim 5.9. *Let $\mathcal{N}_k$ be some k-admissible distribution. Then the following hold.*

1. *For all $x \in \mathbb{Z} \setminus \{0\}$ and $k \in \mathbb{N}$, it holds that*

$$\Pr_{y \xleftarrow{R} \mathcal{N}_k}[\gcd(|x|, y) > 1] \leq \log|x| \cdot 2^{-k} \ .$$

2. *The probability that $y \xleftarrow{R} \mathcal{N}_k$ has a prime factor smaller than $2^{k/2}$ is at most $2^{-k/2}$.*

Proof. For the first part, it holds that x has at most $\log|x|$ prime factors. Since $\mathcal{N}_k$ is admissible, it is divisible by each one of these primes with probability at most 2^{-k}. Using the union bound, the claim follows. For the second part, we apply union bound over all at most $2^{k/2}$ primes smaller than $2^{k/2}$. □

5.2 Proof of the Main Theorems (Thms 5.3 and 5.4)

Recall that our constructions are in canonical form, i.e., they are defined by length function $\ell = \ell(n)$, ring arity $\sigma(n)$, integer-valued vectors $V_n = (\mathbf{v}_1, \ldots, \mathbf{v}_\ell, \mathbf{v}_{\text{zt}})$, and a sampling circuit $\mathcal{D}_\lambda(n, K)$ that given oracle access to a ring, samples $\ell(n)$ ring elements $(a_1, \ldots, a_\ell)$. Therefore, by Lemma 3.7, it suffices to prove that SimpleObf is algebraically secure with respect to strictly compatible circuits and that RobustObf is algebraically secure with respect to compatible circuits. (Recall that compatibility is defined with respect to the level vectors V_n.)

Fix some input length n, security parameter λ, function identifier K and let $\ell = \ell(n)$, $V = V_n$, and $\mathcal{D}_K = \mathcal{D}_\lambda(n, K)$. Let $\mathcal{R}$ be some $(\|\mathbf{v}_{\text{zt}}\|_1 + \omega(\log(\lambda)))$-admissible ring distribution. Fix an adversary A, and recall that A is a polynomial-size purely arithmetic circuit A which is compatible with the corresponding level vectors V.[16] We let P_0 denote the ℓ-variate polynomial induced by A. Since the circuit is V-compatible the degree cannot go above $\|\mathbf{v}_{\text{zt}}\|_1$. Given a black-box access to the function C_K, our goal is to determine (up to negligible error) whether the polynomial P_0 evaluates to zero over the ring elements $(a_i)_{i\in[\ell]}$ which are sampled according to $\mathcal{D}_K^{\mathcal{R}}$. Before analyzing $P_0(\vec{a})$, we make a short detour and take a closer look at the the way $\vec{a}$ is generated by $\mathcal{D}_K$.

In both obfuscators, the circuit $\mathcal{D}_K$ is in *El-Gamal form* (EG in short), namely:

1. It samples a vector of $\ell/2$ uniform ring elements $\vec{r} = (r_1, \ldots, r_{\ell/2})$.

2. Independently of $\vec{r}$, $\mathcal{D}_K$ samples a vector of $\ell/2$ rings elements $\vec{w} = (w_1, \ldots, w_{\ell/2})$. This distribution depends on K and is *non-uniform* over the ring. (In particular, the same w_i may appear twice).

3. The output $(a_i)_{i\in[\ell]}$ consists (up to public re-ordering) $\ell/2$ pairs of the form $(r_i, z_i)_{i\in[\ell/2]}$ where $z_i = r_i \cdot w_i$. (The pair (r_i, z_i) may not be encoded in the same level.)

The polynomial $P_0(\vec{a})$ can be therefore written as a polynomial $P_0(\vec{r}, \vec{z})$ in the formal variables $(\vec{r}, \vec{z})$. Moreover, by substituting $z_i = r_i w_i$ we derive a new polynomial

$$P(\vec{r}, \vec{w}) = P_0(\vec{r}, \{r_i w_i\}_i).$$

There is a one-to-one correspondence between the monomials of the formal polynomial $P_0(\vec{r}, \vec{z})$ and the monomials of the formal polynomial $P(\vec{r}, \vec{w})$, that is implied by the transformation $z_i = r_i w_i$ in one direction, and the converse $z_i = w_i / r_i$ in the other direction. This correspondence extends to any subset of monomials. Moreover, since each r_i and z_i are associated with a level $\mathbf{v}_i$, we can define the level of a monomial μ_0 of P_0 as the sum of levels of the operands of the product. We extend the notion of level of a monomial also to monomials of P via the aforementioned correspondence.

We summarize some of the properties of the polynomial $P(\vec{r}, \vec{w})$.

Lemma 5.10. *The polynomial P induced by P_0 can be written as*

$$P(\vec{r}, \vec{w}) = \sum_j T_j(\vec{r}) Q_j(\vec{w}) \ , \tag{2}$$

where the T_j's are distinct monomials, the Q_j's are general polynomials, and, for every j, i, if T_j is r_i-free then Q_j is w_i-free. Moreover, if the adversary is strictly compatible (resp., weakly compatible) then the levels of all monomials in P is exactly $\mathbf{v}_{zt}$ (resp., at most $\mathbf{v}_{zt}$).

The lemma asserts that if w_i "appears" in Q_j then r_i must divide T_j. This fact will be used later in the following subsections.

Proof. Consider a monomial $\mu(\vec{r}, \vec{w})$ of P (i.e. a product of a subset of $\vec{r}, \vec{w}$ with nonzero coefficient in the expansion of P). We need the following sub-claim: For all i^*, if μ is r_{i^*}-free then it is also w_{i^*}-free. Indeed, since P is obtained from P_0 by substitution of $z_i = r_i w_i$, we can identify a

[16]In the case of SimpleObf the circuit is actually strictly compatible, but this point will be only used later in Section 5.3.

monomial $\mu_0 = \mu_0(\vec{r}, \vec{z})$ in P_0 s.t. substituting $z_i = r_i w_i$ in μ_0 implies μ. Assume that μ is not w_{i^*}-free, then it must be the case the μ_0 is not z_{i^*}-free. Namely, $\mu_0(\vec{r}, \vec{z}) = z_{i^*} \cdot \mu_0'(\vec{r}, \vec{z})$. After substituting $\vec{z}$ to obtain μ, we get $\mu(\vec{r}, \vec{w}) = r_{i^*} w_{i^*} \cdot \mu'(\vec{r}, \vec{w})$ and the claim follows.

Let us now decompose P according to (2). (Observe that such a unique decomposition always exist.) Fix some monomial T in this decomposition. We can write each $T(\vec{r})Q(\vec{w}) = \sum_j c_j \mu_j(\vec{r}, \vec{w})$, where $c_j \neq 0$ and each μ_j is a monomial in $\vec{r}, \vec{w}$. Syntactically it holds that $\mu_j(\vec{r}, \vec{w}) = T(\vec{r}) \cdot \nu_j(\vec{w})$ for some monomials ν_j. Thus if T is r_i-free then all μ_j's are r_i-free, then we can apply the above claim to each μ_j and deduce that they are also w_i-free. We conclude that if T is r_i-free then Q is w_i-free.

Finally, if the adversary is strictly compatible (resp., compatible) then the levels of all monomials in P_0 is exactly $\mathbf{v}_{\mathrm{zt}}$ (resp., at most $\mathbf{v}_{\mathrm{zt}}$). $\square$

Our goal now reduces to testing whether $P(\vec{r}, \vec{w}) = 0$ where the inputs are selected according to $\mathcal{D}_K^{\mathcal{R}}$. We begin by showing that this probabilistic test reduces to an algebraic test. That is, it suffices to check whether a related polynomial equivalent to the zero polynomial over the integers. Before formalizing this statement, we need some definitions.

The polynomials $\mathcal{D}[\![i]\!]$. Recall that $\mathcal{D}_K$ is a circuit (which is not ring-specific) that uses special sampling gates $\{S_i\}_{i \in [\sigma]}$ (in addition to generic ring operations). Each occurrence of the gate S_i samples a uniform element from the ith subring. Therefore, for every subring $i \in [\sigma]$ the circuit $\mathcal{D}_K$ induces as a formal multivariate polynomial $\mathcal{D}[\![i]\!]$ with integer coefficients whose formal variables correspond the occurrences of the gate S_i. Put differently, we view $\mathcal{D}[\![i]\!]$ as the projection of the polynomial computed by $\mathcal{D}_K$ to the i-th subring. We refer to the variables of $\mathcal{D}[\![i]\!]$ as atomic variables. Let $P_{|\mathcal{D}[\![i]\!]}$ be the formal polynomial (with integer coefficients) over the atomic variables obtained by substituting each variable of P with the value assigned to it by $\mathcal{D}_K[\![i]\!]$.

We can now reduce the simulation problem to the task of determining whether the polynomials $P_{|\mathcal{D}[\![1]\!]}, \ldots, P_{|\mathcal{D}[\![\sigma]\!]}$ are the zero polynomials (over the integers).

Lemma 5.11. *Assume the existence of a deterministic simulator $\mathcal{S}$ that, given the circuit P_0 and an oracle access to C_K, outputs 0 if and only if for every subring $i \in [\sigma]$ the polynomial $P_{|\mathcal{D}_K[\![i]\!]}$ is identically zero as a polynomial over the integers. Then, the simulator has a negligible one-sided error. That is,*

- *If $\mathcal{S}$ outputs 0 then* $\Pr[P(\vec{r}, \vec{w}) = 0] = 1$,
- *If $\mathcal{S}$ outputs 1 the* $\Pr[P(\vec{r}, \vec{w}) = 0] = \mathrm{negl}(\lambda)$,

where the probabilities taken over a random choice of a ring $\mathbb{Z}_N \xleftarrow{R} \mathcal{R}$ and a random assignment of the atomic variables.

Proof. The first part is trivial. Clearly, if the integer polynomial $P_{|\mathcal{D}_K[\![i]\!]}$ is zero for every sub-ring $i \in [\sigma]$ then P vanishes for any choice of a ring $\mathbb{Z}_N \xleftarrow{R} \mathcal{R}$ and any assignment for $(\vec{r}, \vec{w})$.

We move on to the second part. Assume that, for some $i \in [\sigma]$, the polynomial $P_{|\mathcal{D}_K[\![i]\!]}$ is not identically zero over the integers. Let $a \neq 0$ be some non-zero coefficient of $P_{|\mathcal{D}_K[\![i]\!]}$. We first claim that $|a|$ is upper-bounded by $2^{\|\mathbf{v}_{\mathrm{zt}}\|_1 \mathrm{poly}(\lambda)}$. To see this, observe that the polynomial $P_{|\mathcal{D}_K[\![i]\!]}$ can be computed by a purely arithmetic circuit of size $\mathrm{poly}(\lambda)$. By Fact 5.7 each of the coefficients of such a polynomial is at most $2^{\|\mathbf{v}_{\mathrm{zt}}\|_1 \mathrm{poly}(\lambda)}$ in absolute value.

Having established an upper-bound on $|a|$, we consider the event where the chosen ring $\mathbb{Z}_N \overset{R}{\leftarrow} \mathcal{R}$ satisfies the following: (1) a is a unit in the i-th subring of $\mathbb{Z}_N$; and (2) the size $N_{\min}$ of the smallest sub-ring of $\mathbb{Z}_N$ is at least $2^{2\log\|\mathbf{v}_{\mathrm{zt}}\|_1+\omega(\log\lambda)} = \|\mathbf{v}_{\mathrm{zt}}\|_1{}^2 \cdot \lambda^{\omega(1)}$. Since the ring is chosen from a $(4\log\|\mathbf{v}_{\mathrm{zt}}\|_1 + \omega(\log\lambda))$-admissible ring distribution $\mathcal{R}$, Claim 5.9 guarantees that the above event happens with all but $\mathrm{negl}(\lambda)$ probability. To see this, for the first part we use the first item in the claim to deduce that the probability of a not being a unit is at most $(\log|a|)\cdot 2^{-(4\log\|\mathbf{v}_{\mathrm{zt}}\|_1+\omega(\log\lambda))} = \mathrm{negl}(\lambda)$. The second part follows straightforwardly from the second item in the claim.[17]

From now on, we condition on the above event. By (1), the polynomial $P^{\mathbb{Z}_N}_{|\mathcal{D}_K[\![i]\!]}$ is still a non-zero polynomial of degree at most $\|\mathbf{v}_{\mathrm{zt}}\|_1$ over the subring $\mathbb{Z}_N[\![i]\!]$. Moreover, we claim that the degree of $P_{|\mathcal{D}_K[\![i]\!]}$ is at most $\|\mathbf{v}_{\mathrm{zt}}\|_1{}^2$. Indeed,

$$\deg(P_{|\mathcal{D}_K[\![i]\!]}) = \deg(P)\cdot\deg(\mathcal{D}_K[\![i]\!]) \le \|\mathbf{v}_{\mathrm{zt}}\|_1 \cdot \|\mathbf{v}_{\mathrm{zt}}\|_1,$$

where the upper-bound on $\deg(P)$ follows from the fact that P is V-compatible and the upper-bound on $\deg(\mathcal{D}_K[\![i]\!])$ is by construction. (In fact, $\deg(\mathcal{D}_K[\![i]\!]) \le n + \deg(\hat{\mathcal{U}}) + 1 \le \|\mathbf{v}_{\mathrm{zt}}\|_1$.)

We can now apply (a variant of) the Schwartz-Zippel lemma (Fact 5.5), and upper-bound the probability that the polynomial $P^{\mathbb{Z}_N}_{|\mathcal{D}_K[\![i]\!]}$ evaluates to zero (over a uniform choice of its atomic inputs) by $\|\mathbf{v}_{\mathrm{zt}}\|_1{}^2/N_{\min} < \mathrm{negl}(\lambda)$. The lemma follows. □

Our goal is now to implement a simulator that tests if $P_{|\mathcal{D}_K[\![i]\!]} \equiv 0$. We use the decomposition (2) to reduce this problem to a zero-testing of the Q_j's. Formally, let Q_j be one of the polynomials defined in (2). As before, we let $Q_{j|\mathcal{D}[\![i]\!]}$ denote the formal polynomial obtained by substituting each variable of Q with the ith subring component of the value assigned to it by $\mathcal{D}_K$.

Claim 5.12. *Fix some subring $i \in [\sigma]$. Then $P_{|\mathcal{D}_K[\![i]\!]} \equiv 0$ if and only if for every polynomial Q_j in the decomposition (2), it holds that $Q_{j|\mathcal{D}[\![i]\!]}$ is the zero polynomial over the integers.*

Proof. The "if" direction is immediate from (2). We prove the "only if" direction. Assume, without loss of generality, that the polynomial $Q_{1|\mathcal{D}_K[\![i]\!]}$ is not identically zero over the integers. Note that the corresponding term $T_1(\vec{r})$ remains unchanged when projected to the ith subring since the $\vec{r}$ variables are atomic variables that are correspond to sample gates from the entire ring. It therefore follows that $P_{|\mathcal{D}_K[\![i]\!]}$ must have a non-zero monomial that contains $T_1(\vec{r})$. The claim follows. □

We conclude that a simulator that tests whether $Q_{j|\mathcal{D}[\![i]\!]} \equiv 0$ for each i and j, has only a negligible error. The following lemma shows that such a test can be implemented given only a black-box access to C_K.

Lemma 5.13 (key lemma)**.** *Both for the case of* SimpleObf *and* RobustObf*, there exists a deterministic algorithm (tester) that given the circuit P_0, an oracle access to C_K and a polynomial Q_j in the decomposition (2), outputs 1 if and only if there exists a subring i for which the polynomial $Q_{j|\mathcal{D}_K[\![i]\!]}$ is not identically zero as a polynomial. The tester makes at most a single query to C_K per Q_j.*

[17]This is the only place in the proof where the admissability of the ring is being used. Correspondingly, admissability can be replaced by any property that guarantees that the coefficient a is a unit in all subrings of $\mathbb{Z}_N$, and that the size of the smallest subring of $\mathbb{Z}_N$ is at least $\|\mathbf{v}_{\mathrm{zt}}\|_1{}^2 \cdot \lambda^{\omega(1)}$ (the latter is for the sake of applying Schwartz-Zippel).

We remark that the tester algorithm will be computationally inefficient since it relies on the decomposition (2) from Lemma 5.10. However, computational efficiency is not required by Definition 3.2.

The proof of the lemma is deferred to Section 5.3 (for the case of SimpleObf) and to Section 5.4 (for the case of RobustObf). This completes the proof of Theorems 5.3 and 5.4. $\square$

5.3 Proof of Lemma 5.13 for the case of SimpleObf

Fix some polynomial $Q = Q_j$ and let $T = T_j$ be the corresponding monomial in the decomposition of P as in (2). In order to prove Lemma 5.13, we will need some structural claims about the polynomial Q.

Claim 5.14. *There exists $x = (x_1, \dots, x_n) \in \{0,1\}^n$ such that Q is $\{w_{i,1-x_i}\}_{i\in[n]}$-free.*

Proof. We show that for all i, T is either $r_{i,0}$-free or it is $r_{i,1}$-free. Combining this with Lemma 5.10, the claim follows.

Let μ be a monomial in the expansion of $T(\vec{r})Q(\vec{w})$ and let μ_0 be the corresponding monomial in P_0. Since we are in the weak GES model, the level of μ_0 is exactly $\mathbf{v}_{\text{zt}}$. Assume towards contradiction that there exists i s.t. T is neither $r_{i,0}$-free nor $r_{i,1}$-free. Then the monomial $\mu_0(\vec{r}, \vec{z})$ contains either $r_{i,0}$ or $z_{i,0}$, and in addition either $r_{i,1}$ or $z_{i,1}$. This implies that the level of μ_0 is equal on one hand to $\mathbf{v}_{\text{zt}}$ and on the other to $\mathbf{v}_{i,0} + \mathbf{v}_{i,1} + \sum_j \mathbf{v}_j$ where $\mathbf{v}_j$ are vectors from the set described in Figure 1.

A combinatorial argument (a la straddling sets) implies that this is not possible. More formally the ith column of $\mathbf{v}_{i,0} + \mathbf{v}_{i,1}$ is $\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}$, and in order to complete it to $\mathbf{v}_{\text{zt}}$ the imbalance between the middle element and the other elements needs to be corrected. Since none of the vectors in Figure 1 can make this correction without going beyond the $M[i]$ value (and thus going above $\mathbf{v}_{\text{zt}}$) the claim follows. $\square$

Claim 5.15. *The polynomial P (and therefore each one of its Q terms) is*

$$\left\{ w_{i,b}^{(M[i]+1)}, w_i^{(M[i]+1)}, w_0^2, \hat{w}_{i,b}\hat{w}_{i,b'} \right\}_{i\in[n], b, b' \in \{0,1\}} \text{-free.}$$

Proof. Using the correspondence between P and P_0, the claim is equivalent to the claim that P_0 is $\{z_{i,b}^{(M[i]+1)}, z_i^{(M[i]+1)}, z_0^2, \hat{z}_{i,b}\hat{z}_{i,b'}\}_{i,b,b'}$-free. We again use the fact that the level of P_0 is $\mathbf{v}_{\text{zt}}$ and consider the level of each of the monomials $\{z_{i,b}^{(M[i]+1)}, z_i^{(M[i]+1)}, z_0^2, \hat{z}_{i,b}\hat{z}_{i,b'}\}_{i,b}$, namely $\{(M[i]+1)\mathbf{v}_{i,b}, (M[i]+1)\mathbf{v}_i, 2\mathbf{v}_0, \hat{\mathbf{v}}_{i,b} + \hat{\mathbf{v}}_{i,b'}\}$ respectively. All of the above, except $\hat{\mathbf{v}}_{i,b} + \hat{\mathbf{v}}_{i,b'}$ when $b \neq b'$ already exceed the $\mathbf{v}_{\text{zt}}$ value in some coordinate and therefore trivially cannot be completed to $\mathbf{v}_{\text{zt}}$. As for $\hat{\mathbf{v}}_{i,0} + \hat{\mathbf{v}}_{i,1}$, it's ith column takes the value $\begin{bmatrix} M[i] \\ 0 \\ M[i] \end{bmatrix}$ and any other $\mathbf{v}$ vector either has no entry in this column and is thus insufficient for reaching $\mathbf{v}_{\text{zt}}$ or if it has nonzero value in this column $(\mathbf{v}_{i,0}, \mathbf{v}_{i,1}, \mathbf{v}_i, \hat{\mathbf{v}}_{i,0}, \hat{\mathbf{v}}_{i,1})$ will cause the level to go above $\mathbf{v}_{\text{zt}}$. $\square$

We can now prove Lemma 5.13.

Recall that we are given a polynomial $Q = Q(\vec{w})$ which is a formal polynomial over the integers in variables $(w_0, \{\hat{w}_{i,b}\}_{i,b}, \{w_{i,b}\}_{i,b}, w_{n+1}, \dots, w_{n+m})$. By Claim 5.14, there exists $x = (x_1, \dots, x_n) \in \{0,1\}^n$ s.t. Q is only a function of $(w_0, \{\hat{w}_{i,b}\}_{i,b}, \{w_{i,x_i}\}_i, w_{n+1}, \dots, w_{n+m})$. We define the polynomials Q', Q'' (with integer coefficients) as follows. The polynomial Q' is identical to Q, only it assigns a single variable $\hat{w}_i$ in the place of both $\hat{w}_{i,0}, \hat{w}_{i,1}$. Therefore Q' is a polynomial in the variables $(w_0, \{\hat{w}_i\}_i, \{w_{i,x_i}\}_i, w_{n+1}, \ldots, w_{n+m})$. We note that this substitution is consistent with $\mathcal{D}_K$, that is $Q'_{|\mathcal{D}_K} = Q_{|\mathcal{D}_K}$. The polynomial Q'' further substitutes

$$w_0 = \Big(\prod_{i\in[n]} \hat{w}_i\Big) \cdot \hat{\mathcal{U}}(w_{1,x_1}, \ldots, w_{n,x_n}, w_{n+1}, \ldots, w_{n+m}) \ .$$

We proceed by case analysis.

1. If Q or Q' are the zero polynomial then the algorithm trivially outputs 0.
2. If Q'' is not the zero polynomial we output 1. Indeed, when Q'' is not the zero polynomial, it must be the case that $Q_{|\mathcal{D}_K[\![1]\!]}$ is not the zero polynomial. This is because $\vec{w}[\![1]\!]$ is a function of an unconstrained set of variables $\vec{y}$, s.t. $w_{i,x_i}[\![1]\!] = y_i$ regardless of x_i. That is

   $$(\hat{w}_1, \ldots, \hat{w}_n, w_{1,x_1}, \ldots, w_{n,x_n}, w_{n+1}, \ldots, w_{n+m})[\![1]\!] = (\hat{y}_1, \ldots, \hat{y}_n, y_1, \ldots, y_n, y_{n+1}, \ldots, y_{n+m}) \ .$$

   Therefore $Q_{|\mathcal{D}_K[\![1]\!]} = Q''(\vec{y})$ and the claim follows.
3. If Q' is not the zero polynomial but Q'' is the zero polynomial, then we output 0 if and only if $C_K(x) = 1$. The rest of the proof is devoted to the analysis of the last case.

Analysis of case 3. If Q' is not the zero polynomial but Q'' is the zero polynomial, it must be the case that

$$\left(w_0 - \Big(\prod_{i\in[n]} \hat{w}_i\Big) \cdot \hat{\mathcal{U}}(w_{1,x_1}, \ldots, w_{n,x_n}, w_{n+1}, \ldots, w_{n+m})\right) | Q'(\vec{w}) \ ,$$

since Q'' is obtained from Q' by assigning a value to w_0 (see Fact 5.6). Furthermore, it holds that

$$Q'(\vec{w}) = a \cdot \left(w_0 - \Big(\prod_{i\in[n]} \hat{w}_i\Big) \cdot \hat{\mathcal{U}}(w_{1,x_1}, \ldots, w_{n,x_n}, w_{n+1}, \ldots, w_{n+m})\right) \ ,$$

for some non-zero constant a, since by Claim 5.15, the degree of all variables in Q' cannot be higher than their degree in the expression on the right hand side. Since $Q_{|\mathcal{D}_K[\![1]\!]} = Q''(\vec{y}) = 0$, it suffices to check whether $Q_{|\mathcal{D}_K[\![2]\!]} = Q'_{|\mathcal{D}_K[\![2]\!]}$ is the zero polynomial. In the second sub-ring, it holds that

$$(\hat{w}_1, \ldots, \hat{w}_n, w_{1,x_1}, \ldots, w_{n,x_n}, w_{n+1}, \ldots, w_{n+m})[\![2]\!] = (\hat{\beta}_1, \ldots, \hat{\beta}_n, x_1, \ldots, x_n, K_1, \ldots, K_m) \ ,$$

and in addition

$$w_0[\![2]\!] = \prod_{i\in[n]} \hat{\beta}_i \ ,$$

where $\hat{\beta}$ are uniform variables.

It follows that

$$\begin{aligned} Q_{|\mathcal{D}_K[\![2]\!]} &= a \cdot \prod_{i\in[n]} \hat{\beta}_i \cdot (1 - \hat{\mathcal{U}}(x_1, \ldots, x_n, K_1, \ldots, K_m)) \\ &= a[\![2]\!] \cdot \prod_{i\in[n]} \hat{\beta}_i \cdot (1 - C_K(x)) \ . \end{aligned}$$

Therefore $Q_{|\mathcal{D}_K[\![2]\!]}$ is identically zero if and only if $C_K(x) = 1$. The lemma follows. □

5.4 Proof of Lemma 5.13 for the case of RobustObf

Fix some polynomial $Q = Q_j$ and let $T = T_j$ be the corresponding monomial in the decomposition of P as in (2). In order to prove Lemma 5.13, we will need some structural claims about the polynomial Q.

Claim 5.16. *The polynomial Q is $(w_0 w)$-free for all w in $(w_0, \{\hat{w}_{i,b}\}_{i,b}, \{w_{i,b}\}_{i,b}, w_{n+1}, \dots, w_{n+m})$.*

Proof. We again refer to the notion of level. The claim in the lemma is equivalent to requiring that the polynomial $P_0(\vec{r}, \vec{z})$ is $(z_0 z)$-free for all z. Furthermore, the model enforces that the level of P_0 is $\le \mathbf{v}_{\mathrm{zt}}$. We notice that the level of z_0 is $\ge D\mathbf{v}^*$, and the level of the other z variables is $\ge \mathbf{v}^*$. It follows that any product of z_0 and another z variable will be at level $\ge (D+1)\mathbf{v}^*$. Since $(D+1) \not\le \mathbf{v}_{\mathrm{zt}}$, such term cannot appear in P_0 and therefore the claim about Q follows. ☐

Claim 5.17. *For all $i \in [n]$, $b_1, b_2 \in \{0,1\}$ the polynomial Q is $(\hat{w}_{i,b_1} \hat{w}_{i,b_2})$-free.*

Proof. We again turn to the level of P_0 and use it to show that it must be $(\hat{z}_{i,b_1} \hat{z}_{i,b_2})$-free. This holds since $(\hat{z}_{i,b_1} \hat{z}_{i,b_2})$ has level $\hat{\mathbf{v}}_{i,b_1} + \hat{\mathbf{v}}_{i,b_2}$, which, for any values of b_1, b_2, is $\not\le \mathbf{v}_{\mathrm{zt}}$. ☐

Claim 5.18. *For all $i \in [n]$, $b \in \{0,1\}$ the polynomial Q is $(\hat{w}_{i,b_1} w_{i,0} w_{i,1})$-free.*

Proof. Similarly to above, we prove that P_0 is $(\hat{z}_{i,b_1} z_{i,0} z_{i,1})$-free. This holds since $(\hat{z}_{i,b_1} z_{i,0} z_{i,1})$ has level $\hat{\mathbf{v}}_{i,b} + \mathbf{v}_{i,0} + \mathbf{v}_{i,1} \not\le \mathbf{v}_{\mathrm{zt}}$. ☐

We can now prove Lemma 5.13.

Similarly to the proof for SimpleObf, we define some auxiliary polynomials. We are given a polynomial $Q = Q(\vec{w})$ which is a formal polynomial over the integers in variables

$$(w_0, \{\hat{w}_{i,b}\}_{i,b}, \{w_{i,b}\}_{i,b}, w_{n+1}, \dots, w_{n+m}) \ .$$

We define the polynomial Q' which is identical to Q, only it assigns a single variable $\hat{w}_i$ in the place of both $\hat{w}_{i,0}, \hat{w}_{i,1}$. Therefore Q' is a polynomial in the variables:

$$(w_0, \{\hat{w}_i\}_i, \{w_{i,b}\}_{i,b}, w_{n+1}, \dots, w_{n+m}) \ .$$

This substitution is consistent with $\mathcal{D}_K$, that is $Q'_{|\mathcal{D}_K} = Q_{|\mathcal{D}_K}$ (over any subring). We then define $\hat{Q}''$ to be the polynomial that is identical to Q' except it substitutes $w_0 = 0$, and $\hat{Q}''_i$ which in addition sets $\hat{w}_i = 0$. Finally for all $x \in \{0,1\}^n$ we define Q''_x to be the polynomial that is identical to Q' except for substituting

$$w_0 = \Big(\prod_{i \in [n]} \hat{w}_i\Big) \cdot \hat{\mathcal{U}}(w_{1,x_1}, \dots, w_{n,x_n}, w_{n+1}, \dots, w_{n+m}) \ .$$

Note that $\hat{Q}''$ and Q''_x each makes different substitutions to the value of w_0 in Q'.

We again proceed with case analysis.

1. If $\hat{Q}''_i \not\equiv 0$ for some i, then return 1. In this case it must be that $Q_{\mathcal{D}_K[\![i+2]\!]} \not\equiv 0$. To see this, observe that $\vec{w}_{|\mathcal{D}_K[\![i+2]\!]}$ contains independent ρ variables for all but $w_0, \hat{w}_i$ for which a zero value is assigned. Thus $\hat{Q}''_i$ is exactly the same polynomial as $Q_{\mathcal{D}_K[\![i+2]\!]}$ up to renaming of variables.

We now note that if $\hat{Q}''_i \equiv 0$ for every i, then this means that $\prod_i \hat{w}_i | \hat{Q}''$. This means, by Claim 5.18, that there exists $x = (x_1, \ldots, x_n) \in \{0,1\}^n$ s.t. $\hat{Q}''$ is $w_{i,1-x_i}$-free. This is since otherwise $\hat{Q}''$ and therefore Q' are not $(\hat{w}_i w_{i,0} w_{i,1})$-free for some i, which implies that Q is not $(\hat{w}_{i,b} w_{i,0} w_{i,1})$-free contrary to the claim. Let us fix this x for the remainder of the algorithm and proof.

Therefore, considering Claim 5.16 and Claim 5.17, in such case, we can write Q' as

$$Q'(w_0, \hat{w}_1, \ldots, \hat{w}_n, w_{1,x_1}, \ldots, w_{n,x_n}, w_{n+1}, \ldots, w_{n+m}) = \\ a w_0 + (\prod_{i\in[n]} \hat{w}_i) Q'''(w_{1,x_1}, \ldots, w_{n,x_n}, w_{n+1}, \ldots, w_{n+m}) \ , \quad (3)$$

for some constant a and for the specific x described above.

2. Continuing to consider the case where $\hat{Q}''_i \equiv 0$ for every i, we now consider the case where in addition $Q''_x \not\equiv 0$ for the x that we defined above, then return 1. Indeed, in this case $Q'_{|\mathcal{D}_K[\![1]\!]} \not\equiv 0$ (and therefore also $Q_{|\mathcal{D}_K[\![1]\!]} \not\equiv 0$) since

$$(\hat{w}_1, \ldots, \hat{w}_n, w_{1,x_1}, \ldots, w_{n,x_n}, w_{n+1}, \ldots, w_{n+m})[\![1]\!] = (\hat{y}_1, \ldots, \hat{y}_n, y_1, \ldots, y_n, y_{n+1}, \ldots, y_{n+m}) \ ,$$

and in addition
$$w_0[\![1]\!] = (\prod_{i\in[n]} \hat{y}_i) \cdot \hat{\mathcal{U}}(y_1, \ldots, y_{n+m}) \ ,$$

where the y variables are unconstrained.

Therefore in this case Q''_x is identical to $Q'_{|\mathcal{D}_K[\![1]\!]}$ up to renaming variables, and so the latter polynomial is not identically zero, as claimed.

3. Finally, if for all i it holds that $\hat{Q}''_i \equiv 0$, and in addition $Q''_x \equiv 0$ for the x defined above, then we output 0 if and only if $C_K(x) = 1$. The rest of the proof is devoted to the analysis of this case.

Analysis of case 3. We recall Eq. (3) and deduce that

$$Q'(\vec{w}) = a \cdot \left(w_0 - (\prod_{i\in[n]} \hat{w}_i) \cdot \hat{\mathcal{U}}(w_{1,x_1}, \ldots, w_{n,x_n}, w_{n+1}, \ldots, w_{n+m}) \right) \ .$$

Similarly to the analysis of $\mathsf{SimpleObf}$, we consider the last remaining subring: $Q_{|\mathcal{D}_K[\![2]\!]} = Q'_{|\mathcal{D}_K[\![2]\!]}$, where it holds that

$$(\hat{w}_1, \ldots, \hat{w}_n, w_{1,x_1}, \ldots, w_{n,x_n}, w_{n+1}, \ldots, w_{n+m})[\![2]\!] = (\hat{\beta}_1, \ldots, \hat{\beta}_n, x_1, \ldots, x_n, K_1, \ldots, K_m) \ ,$$

and in addition
$$w_0[\![2]\!] = \prod_{i\in[n]} \hat{\beta}_i \ ,$$

where $\hat{\beta}$ are unconstrained variables. It follows that

$$\begin{aligned} Q_{|\mathcal{D}_K[\![2]\!]} &= a \cdot \prod_{i\in[n]} \hat{\beta}_i \cdot (1 - \hat{\mathcal{U}}(x_1, \ldots, x_n, K_1, \ldots, K_m)) \\ &= a[\![2]\!] \cdot \prod_{i\in[n]} \hat{\beta}_i \cdot (1 - C_K(x)) \ . \end{aligned}$$

Therefore, $Q_{|\mathcal{D}_K[\![2]\!]}$ is identically zero, if and only if $C_K(x) = 1$, and Lemma 5.13 follows. $\square$

References

[AGIS14] Prabhanjan Ananth, Divya Gupta, Yuval Ishai, and Amit Sahai. Optimizing obfuscation: Avoiding barrington's theorem. Cryptology ePrint Archive, Report 2014/222, 2014. `http://eprint.iacr.org/`.

[App14] Benny Applebaum. Bootstrapping obfuscators via fast pseudorandom functions. In *Advances in Cryptology - ASIACRYPT 2014*, pages 162–172, 2014.

[BD16] Zvika Brakerski and Or Dagmi. Shorter circuit obfuscation in challenging security models. In Vassilis Zikas and Roberto De Prisco, editors, *Security and Cryptography for Networks - 10th International Conference, SCN 2016, Amalfi, Italy, August 31 - September 2, 2016, Proceedings*, volume 9841 of *Lecture Notes in Computer Science*, pages 551–570. Springer, 2016.

[BGI$^+$12] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2):6, 2012. Preliminary version in CRYPTO 2001.

[BGK$^+$14] Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Protecting obfuscation against algebraic attacks. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 221–238. Springer, 2014.

[BR13] Zvika Brakerski and Guy N. Rothblum. Obfuscating conjunctions. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 416–434. Springer, 2013.

[BR14a] Zvika Brakerski and Guy N. Rothblum. Black-box obfuscation for d-cnfs. In Moni Naor, editor, *Innovations in Theoretical Computer Science, ITCS'14, Princeton, NJ, USA, January 12-14, 2014*, pages 235–250. ACM, 2014.

[BR14b] Zvika Brakerski and Guy N. Rothblum. Virtual black-box obfuscation for all circuits via generic graded encoding. In Yehuda Lindell, editor, *Theory of Cryptography - 11th Theory of Cryptography Conference, TCC 2014, San Diego, CA, USA, February 24-26, 2014. Proceedings*, volume 8349 of *Lecture Notes in Computer Science*, pages 1–25. Springer, 2014.

[BWZ14] Dan Boneh, David J. Wu, and Joe Zimmerman. Immunizing multilinear maps against zeroizing attacks. *IACR Cryptol. ePrint Arch.*, 2014:930, 2014.

[CLT13] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Practical multilinear maps over the integers. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 476–493. Springer, 2013.

[GGH13a] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, volume 7881 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2013.

[GGH$^+$13b] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 40–49. IEEE Computer Society, 2013.

[GIS$^+$10] Vipul Goyal, Yuval Ishai, Amit Sahai, Ramarathnam Venkatesan, and Akshay Wadia. Founding cryptography on tamper-proof hardware tokens. In Daniele Micciancio, editor, *Theory of Cryptography, 7th Theory of Cryptography Conference, TCC 2010, Zurich, Switzerland, February 9-11, 2010. Proceedings*, volume 5978 of *Lecture Notes in Computer Science*, pages 308–326. Springer, 2010.

[GLSW14] Craig Gentry, Allison B. Lewko, Amit Sahai, and Brent Waters. Indistinguishability obfuscation from the multilinear subgroup elimination assumption. *IACR Cryptology ePrint Archive*, 2014:309, 2014.

[GLW14] Craig Gentry, Allison B. Lewko, and Brent Waters. Witness encryption from instance independent assumptions. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 426–443. Springer, 2014. References are respective to the ePrint version: `https://eprint.iacr.org/2014/273`.

[Lin16] Huijia Lin. Indistinguishability obfuscation from constant-degree graded encoding schemes. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I*, volume 9665 of *Lecture Notes in Computer Science*, pages 28–57. Springer, 2016.

[Mau05] Ueli . Maurer. Abstract models of computation in cryptography. In Nigel P. Smart, editor, *IMA Int. Conf.*, volume 3796 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2005.

[PM76] Franco P. Preparata and David E. Muller. Efficient parallel evaluation of boolean expression. *IEEE Trans. Computers*, 25(5):548–549, 1976.

[PST14] Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation from semantically-secure multilinear encodings. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 500–517. Springer, 2014.

[Sch80] Jack Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(2):701–717, 1980.

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *EUROCRYPT*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266. Springer, 1997.

[Zim14] Joe Zimmerman. How to obfuscate programs directly. Cryptology ePrint Archive, Report 2014/776, 2014. Appeared in Eurocrypt 2015 [Zim15].

[Zim15] Joe Zimmerman. How to obfuscate programs directly. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 439–467. Springer, 2015.

[Zip79] Richard Zippel. Probabilistic algorithms for sparse polynomials. In *International Symposiumon on Symbolic and Algebraic Computation*, pages 216–226, 1979.