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 same functionality as C_K while hiding all other information about C_K. The first property is formalized via the existence of an efficient universal 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 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

[*]A preliminary version of this paper appears in the proceedings of TCC 2005.

[†]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 ISF grant 468/14 and by an Alon Young Faculty Fellowship.

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".

Somewhat mysteriously, 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) grow with the *formula size* or *branching program* size of the obfuscated program, which are polynomially larger than the *circuit size*. 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 obfuscate a function family $\mathcal{C}$ with complexity which 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 the 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 $\mathcal{C}$. This falls short of a full answer to the above question since in current GES candidates, 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

[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".

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 distinct indices as long as the union of their indices is still a subset of $\mathbf{v}_{\mathrm{zt}}$, 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., $\mathsf{isZero}([g]_{\mathbf{v}_{\mathrm{zt}}}) = 1 \Leftrightarrow g = 0$. In this model we prove the following theorem:

Theorem 1.1. *There exists an indistinguishability obfuscator* SimpleObf *with respect to $\mathcal{MRG}$ for any circuit family $\mathcal{C}$ in $\mathcal{NC}^1$. Moreover, 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 following hold:*

- *(Size) The obfuscated program contains $4n + 2m + 2$ ring elements.*
- *(Evaluation complexity) The complexity of the evaluation algorithm is $O(t)$, and it can be represented as an $O(t)$-size arithmetic circuit 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 large co-primes (whose bit length is polynomial in the security parameter) and the L_1 norm of the zero-testing level $\mathbf{v}_{zt}$ is upper-bounded by 2^d.*[2]

Remark 1.2. *The above parameters are essentially optimal (up to the dependency in the element size of the GES and the security parameter). 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 $\mathcal{C}$.*

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 $\mathbf{v} \subsetneq \mathbf{v}_{\mathrm{zt}}$. 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 each level) cannot be used as it trivially permits low-level zero-testing.[3] 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.[4] 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

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).

[2]The L_1 norm of $\mathbf{v}_{\mathrm{zt}}$ essentially corresponds to the (maximal) total degree of polynomials that can be evaluated on the GES elements. The upper-bound of 2^d can be replaced with the more refined bound of the total degree of the universal evaluation algorithm. All known instantiations of graded encoding schemes require that $\|\mathbf{v}_{\mathrm{zt}}\|_1$ is polynomial in the security parameter, hence the importance of bounding this parameter.

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

[4]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 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].)

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. In this model, we prove the following result.[5]

Theorem 1.3. *There exists an indistinguishability obfuscator* RobustObf *with respect to* $\mathcal{URG}$ *for any circuit family* $\mathcal{C}$ *in* $\mathcal{NC}^1$*, where the* $\mathcal{URG}$ *oracle is defined over an* $(n+2)$*-composite order ring* $\mathcal{R} = \mathbb{Z}_{p_1} \times \cdots \times \mathbb{Z}_{p_{n+2}}$*. Furthermore, the size, evaluation complexity and the* L_1 *norm of the zero-testing level are exactly as in Theorem 1.1.*

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.

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 explicit in [BR14b, BGK+14, AGIS14]). Roughly speaking, a GES-based obfuscator is in canonical form, if given a program identifier K, it samples a tuple $a = (a_1, \ldots, a_\ell)$ from the underlying ring $\mathcal{R}$, and then outputs a GES-encoding of these elements under the labels $(\mathbf{v}_1, \ldots, \mathbf{v}_\ell)$ 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 $a = (a_1, \ldots, a_\ell)$ 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.

Abstracting previous works, we show that algebraic security implies (standard) security relative to an ideal GES oracles.[6] 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 queries. Correspondingly, 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.

[5]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.

[6]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.

1.2 Our Techniques

To illustrate our techniques let us consider the following simplified physical model. The obfuscator is allowed to put ring elements in locked boxes (marked by multisets) and everyone can add/multiply boxes at most T times. 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's bits $K_1, \ldots, K_m$ in m separate boxes (labeled by $\mathbf{v}_1, \ldots, \mathbf{v}_m$), and prepare for each input x_i a pair of boxes (labeled by $\mathbf{v}_{i,0}, \mathbf{v}_{i,1}$) with the values 0 and 1. Given these boxes and an input x, the evaluator can choose the boxes $\mathbf{v}_{i,x_i}$, propagate the values according to the (arithmetization of the) universal circuit $\hat{\mathcal{U}}(\cdot,\cdot)$ and obtain a box which 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*. Assume that the each box has two slots. We keep the second slot untouched as in the previous obfuscator, and fill the first slot with $n + m$ random authenticators $y_1, \ldots, y_{n+m}$ where 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 $\mathbf{v}_0$) that contains the pair $(0, y_0)$ where $y_0 = \hat{\mathcal{U}}(y_1, \ldots, y_{n+m})$, and let us increase the bound 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 $(\hat{\mathcal{U}}(x, K), \hat{\mathcal{U}}(y))$, 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. Specifically, if the adversary apply some $(n + m + 1)$-variate polynomial $F(V_0, (V_1, \ldots, V_m), (V'_1, \ldots, V'_n)) \neq \left(\hat{\mathcal{U}}((V'_1, \ldots, V'_n), (V_1, \ldots, V_m)) - V_0\right)$ 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 $y_0 = \hat{\mathcal{U}}(y_1, \ldots, y_{n+m})$. Then the polynomial F simplifies to a non-trivial low-degree polynomial over the random values $y_1, \ldots, y_{n+m}$ and therefore (by Schwartz-Zippel) vanishes with negligible probability.[7]

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. Note that there are still several technicalities. First, unlike the simplified boxes model, in the GES setting, addition can be applied only over 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, cf. [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.

In addition, the resulting construction is vulnerable to "input-mixing" attacks in which the adversary 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* similarly to [BGK+14]. Roughly speaking, straddling sets force consistency by making sure that input-mixing attacks cannot reach to the top (zero-testing) level. These modifications eventually lead to our basic obfuscator SimpleObf.

[7]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.

Interestingly, SimpleObf is completely broken if low-level zero-testing is allowed. Recall that y_i is shared among the zero and one encoding of the i-th input. Therefore, one can zero-out y_i in a low-level encoding by subtracting their El-Gamal encodings, thus obtaining an element that has zero in one of the slot. At this point one can zero-out all authenticators as well, and fully recover the string 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 $\hat{w}_i$ 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.

Alas, we recall that functionality needs to hold as well. The element w_0, which generalizes the y_0 that we had before, now must have 0 in all of the new slots, since 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 the construction and Section 5 for the proof of security.

Remark 1.4 (The degree restriction)**.** *Due to noise issues, current instantiations of GES only support poly(λ)-multiplicative degree. In particular, the representation length of each element is proportional to the degree. In our context, this restriction translates to a degree restriction on the universal circuit $\mathcal{U}$.*[8] *For the (typical) case of balanced circuits, this results in a logarithmic-depth restriction.*

1.3 Related Works

Ananth et al. [AGIS14] 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 s with obfuscated program size and complexity almost as low as $O(s)$. This is in comparison to

[8]For our purposes, the degree of a boolean circuit is its formula size.

previous methods that used Barrington's theorem and achieved $O(s^2)$ for balanced formulae or $O(s^{3.64})$ for unbalanced. Still, their complexity measure remained the *formula size* of the function family, whereas in this work we show that one can obfuscate relative to the *circuit size* of the family which may be smaller. On the flip side, we use composite order graded encoding schemes that are even newer and less substantiated (and possibly less efficient) than standard prime-order graded encoding schemes.

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 $\mathsf{SimpleObf}$. Zimmerman also presents applications for this new obfuscation method for circuits. Security is proven in a generic model where zero testing below the last level is impossible, similar to our $\mathcal{MRG}$ oracle. Both the obfuscator from [Zim14] and our $\mathsf{SimpleObf}$ are completely broken in a more challenging model where it is possible to test for zero at low levels. Our second obfuscator $\mathsf{RobustObf}$ 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, 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.

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 on vectors $\mathbb{N}^\tau$ which corresponds to inclusion. 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 distinct 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. The definition is 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 $u_1 = [\alpha_1]_{\mathbf{v}}$, $u_2 = [\alpha_2]_{\mathbf{v}}$:*

$$u_1 + u_2 = [\alpha_1 + \alpha_2]_{\mathbf{v}} \quad \text{and} \quad u_1 - u_2 = [\alpha_1 - \alpha_2]_{\mathbf{v}} \ ,$$

   where $\alpha_1 + \alpha_2$ and $\alpha_1 - \alpha_2$ are addition and subtraction in $\mathcal{R}$.

3. *There is an associative binary operation "$\times$" 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 definitions for the prime order definitions.

Definition 2.2 (Efficient Procedures for Graded Encoding Scheme). *We consider a graded encoding schemes (see above) where the following procedures are efficiently computable.*

- *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 $N = \prod p_i$.*[9]

  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.

  We note that for known GES candidates, the running time of the setup procedure (and all other procedures) scales with $\|\mathbf{v}_{zt}\|_1$, and hence we require that this value is provided in unary

- *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 nearly uniform in* p_{i^*}, *and for all* $i \neq i^*$ *it holds that* $a_i = 0$. *We stress it is very important for the security of our constructions that evparams does not enable such functionality.*

- *Encode and Re-Randomize:* $\mathsf{encRand}(params, i, a)$ *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.*

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 $poly(\|\mathbf{v}_{\mathrm{zt}}\|_1)$ time, as our function interface compels. This will be sufficient for our purposes and we therefore ignore noise management throughout this manuscript.

Remark 2.3. *Given params, we can use* $\mathsf{subsamp}$ *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 applied to a set of uniformly distributed variables. These variables may not be shared across CRT components, but they can be shared between elements. E.g. in a 2-composite GES, one can generate* $[((a_1 + a_2) \cdot a_3, b_1)]_{\mathbf{0}}$, $[(a_3 + a_4, b_2)]_{\mathbf{0}}$, $[(a_1 \cdot a_2, b_1 + b_2)]_{\mathbf{0}}$ *(but cannot generate in addition* $[(b_1, a_1)]_{\mathbf{0}}$. *(Note that the product of level zero-encoding results in a level zero encoding.) 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 2.4. *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 these encodings need to appear the same as encodings of non-zero elements. This in turn means that we must forbid the adversary to run* $\mathsf{samp}, \mathsf{encRand}$ *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 subring 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 2.4. Overall, to the best of our knowledge, this candidate is consistent with the requirements of our obfuscator (although we prove security only in a generic model and not under explicit assumptions).

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 is independent of the element itself, and so the adversary can manipulate elements only via the GES oracle. 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.

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 $\mathbf{v} \le \mathbf{v}_{\mathrm{zt}}$ by a string $(\mathbf{v}, \rho)$ where ρ is random string of length $t = (\log|\mathcal{R}| \cdot \lambda)$. The oracle also releases random private/public parameters $evparams, secparams \in \{0,1\}^\lambda$ which are associated with this encoding. From now on, the oracle supports all the GES-operations with resect 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 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 2^λ 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, $\mathcal{MRG}$ generates an element a from $\mathcal{R}$ or the appropriate sub-ring, a uniform length t label, but it 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 $(\mathbf{v}, a, poly(\vec{X}), \mathsf{label}_{\mathbf{v},a,poly(\vec{X})})$, and labels will be unique if the respective formal polynomials are distinct. 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) 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 decouple the computational properties of the GES from the information-theoretic properties of the obfuscator. Indeed, the security of our obfuscator will be established using the algebraic definition.

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, where for every $K \in \{0,1\}^{m(n)}$ the function C_K operates on inputs of length n. 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 *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 computed by polynomial-size circuits of logarithmic depth.[10]

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 $C_K \in \mathcal{C}$, an 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 GES implementation.

Definition 3.1 (Preserving Functionality). *A GES-based obfuscation scheme* ($\mathsf{Obf}, \mathsf{Eval}$) *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* $K \in \{0,1\}^{m(n)}$, *and every* $x \in \{0,1\}^n$, *with all but* $negl(\lambda)$ *probability over the coins of* $\mathsf{Obf}, \mathsf{Eval}$ *and the GES oracle* $\mathcal{G}$ *it holds that:*

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

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* $\mathcal{A}$, *there exists a (computationally unbounded) simulator* $\mathcal{S}$, *such that for every* $n \in \mathbb{N}$ *and* for every $C_K \in \mathcal{C}$ *where* $K \in \{0,1\}^{m(n)}$*:*

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

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

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

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 the random GES model above. This model and the equivalence are implicit in previous works. As before, we let $\mathcal{R}_{\lambda,\sigma,\mathbf{v}_{\text{zt}}}$ be some ensemble of probability distributions over rings.

Definition 3.3 (Obfuscator in Canonical form)**.** *An 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 $\mathbf{v}_1, \dots, \mathbf{v}_\ell$, a zero-testing vector $\mathbf{v}_{zt}$ and a ring arity $\sigma \in \mathbb{N}$.*

2. *Based on λ, K, n, the obfuscator defines a joint distribution $\mathcal{D}_\lambda(n,K)$ over ℓ (generic) ring elements $(a_1, \dots, a_\ell)$.*[11]

3. *Then, the obfuscator initializes the GES which samples $\mathcal{R} \overset{R}{\leftarrow} \mathcal{R}_{\lambda,\sigma,\mathbf{v}_{zt}}$ the obfuscator samples the tuple $(a_1, \dots, a_\ell)$ from $\mathcal{R}$ according to the distribution $\mathcal{D}_\lambda(n,K)$, and outputs the vector of encodings $([a_1]_{\mathbf{v}_1}, \dots, [a_\ell]_{\mathbf{v}_\ell})$ 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 $V_n = (\mathbf{v}_1, \dots, \mathbf{v}_\ell, \mathbf{v}_{zt})$, the distribution $\mathcal{D}_\lambda(n,K)$, and the ring distribution $\mathcal{R}_{\lambda,\sigma,\mathbf{v}_{zt}}$.

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 an adversary A (i.e., some arithmetic circuits) evaluates to zero on $\mathcal{D}_\lambda(n,K)$.

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 $\mathcal{P}_{A,\mathcal{R}} \in \mathcal{R}[X_1, \dots, X_\ell]$ 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.*

[11]More precisely, the distribution is defined by randomized arithmetic circuits with GES oracle-gates to the underlying ring, as explained in Remark 2.3.

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 $(\mathbf{v}_1, \ldots, \mathbf{v}_\ell)$ 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 $V = ((\mathbf{v}_1, \ldots, \mathbf{v}_\ell), \mathbf{v}_{zt})$ if the computation $A(\mathbf{v}_1, \ldots, \mathbf{v}_\ell)$ 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., $\mathbf{v} \leq \mathbf{v}_{zt}$. When $\mathbf{v} = \mathbf{v}_{zt}$ we say that A is* strongly compatible *with V.*

We can now define a simulation-based definition of security which is parameterized by some family of 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 $(\ell, \sigma, V, \mathcal{D}, \mathcal{R}_{\lambda,\sigma,\mathbf{v}_{zt}})$ 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*

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

By default, we consider security against the class of all poly(λ, n)-size purely arithmetic circuits $\mathcal{A} = \{\mathcal{A}_{\lambda,n}\}$ which are V_n-compatible (resp., strongly V_n-compatible) and refer to this notion as algebraic security *(resp.,* strong 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 provides stronger security. Note that the class of V_n-compatible adversaries is strictly larger than the class of *strongly* V_n-compatible, and so security against the former strictly implies security against the latter.

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. *If a canonical GES-based obfuscator $(\ell, \sigma, V, \mathcal{D}, \mathcal{R})$ is algebraically secure (resp., strong algebraically secure) then it is a secure indistinguishably obfuscator relative to the GES oracle $\mathcal{URG}$ (resp., $\mathcal{MRG}$) over the ring distribution $\mathcal{R}$. Furthermore, if the above holds with efficient simulation then the conclusion is strengthened to VBB security in the corresponding model.*

Since the proof of the lemma is implicit in previous works, we only sketch it here for the sake of completeness.

Proof Sketch. 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 $\mathcal{S}_{\text{alg}}^{C_K}$ promised by Definition 3.6. Details follow.

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 $t = (\log |\mathcal{R}_{\lambda,\sigma,\mathbf{v}_{\text{zt}}}| \cdot \lambda)$. 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 initialize the adversary B with the values $\hat{C} = ((\mathbf{v}_i, \rho_i)_{i\in[\ell]}, evparams)$. 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 table and that the operation respects V (i.e., $\mathbf{v} = \mathbf{v}' \leq \mathbf{v}_{\text{zt}}$). If the verification fails we return $\perp$. Otherwise, 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 $(\mathbf{v}, \rho_0, A_0)$, 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 $\mathcal{S}_{\text{alg}}^{C_K}$ on the circuit $(A + A') - A_0$. If the answer is positive return the value $(\mathbf{v}, \rho_0)$; otherwise, if the answer is negative for all tuples on the list, return $(\mathbf{v}, \rho)$ where ρ is a new random string, and add to the list a new triple $(\mathbf{v}, \rho, A + A')$.

Multiplication and negation queries are performed similarly with the natural modifications (e.g., in the case of multiplication one has to verify that $(\mathbf{v} + \mathbf{v}') \leq \mathbf{v}_{\text{zt}}$, 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 $(\mathbf{v}_{\text{zt}}, \rho)$ is handled directly by querying the the algebraic simulator $\mathcal{S}_{\text{alg}}^{C_K}$ on the circuit A which appears in the tuple $(\mathbf{v}_{\text{zt}}, \rho, A)$ (again, if such a tuple does not appear in the list we may answer $\perp$).

It is not hard to verify 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. Furthermore, notice that if the algebraic simulator is efficient then so is the simulator $\mathcal{S}$.

Let us (briefly) move to the case of algebraic security with respect to *strongly* 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), as in this case it suffices to check whether A and B are *formally-equivalent*. Such a check can be done by standard techniques with negligible error. (E.g., choose a sufficiently large prime p of bit length *poly*(λ) and check whether A and B agree on a random vector $z \overset{R}{\leftarrow} \mathbb{Z}_p^\ell$). In fact, such a formal-equivalence check should replace all the calls to the algebraic simulator (even if the output of A and B is in the $\mathbf{v}_{\text{zt}}$ level), except for the ones which correspond to zero-test calls of the adversary B. $\square$

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

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

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

Figure 1: The level vectors for obfuscator SimpleObf.

4 Description of the Obfuscator and Correctness

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 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 let $D_{\hat{\mathcal{U}}}$ 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 x, K inputs. 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 $M_i = \mathbf{s}_{\text{out}}[i]$, 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

For all $i \in [n]$, $b \in \{0,1\}$, we define $\mathbf{v}_{i,b} \in \mathbb{Z}^{(n+m+1)\times 3}$ as $\mathbf{v}_{i,b} = \mathbf{e}_i \otimes [b, 1, 1-b]$. 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 $\mathbf{v}_{\text{zt}} = (\mathbf{s}_{\text{out}} + \mathbf{e}_{n+m+1}) \otimes [1,1,1] \in \mathbb{Z}^{(n+m+1)\times 3}$. We note that for all $x \in \{0,1\}^n$ it holds that $\mathbf{v}_{\text{zt}} = \mathbf{v}_0 + \sum_{i=1}^{n}(M[i]\cdot \mathbf{v}_{i,x_i} + \hat{\mathbf{v}}_{i,x_i}) + \sum_{i=n+1}^{n+m} M[i]\cdot \mathbf{v}_i$.

We illustrate the various level vectors in Figure 1.

Obfuscator SimpleObf:

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

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.

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 $\hat{Z}_{i,b} = [\hat{r}_{i,b} \cdot \hat{w}_i]_{\hat{\mathbf{v}}_{i,b}}$, where $\hat{w}_i = (\hat{y}_i, \hat{\beta}_i)$ are uniform.

4. For all $i \in \{n+1, \dots, n+m\}$, 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 $y_0 = \hat{\mathcal{U}}(y_1, \dots, y_{n+m})$.

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 $i \in \{n+1, \dots, n+m\}$ the elements R_i, Z_i.
   - The elements R_0, Z_0.

We note that all of the required encodings can be efficiently generated using *params*, as explained in Remark 2.3.

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 multiplication gate have the same degree (as polynomials in the input variables and program description). This is straightforward to achieve by adding the constant 1 as one of the elements of the program description, and multiplying by this variable (raised to the proper degree) to balance the input degrees.

For all $i \in [n]$, $b \in \{0,1\}$, we define $\mathbf{v}_{i,b} \in \mathbb{Z}^{(n+m+1)\times 4}$ 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 $\mathbf{v}_i = \mathbf{e}_i \otimes [1,1,1,1]$. 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 $\mathbf{v}_{\mathrm{zt}} = (\mathbf{s}_{\mathrm{out}} + \mathbf{e}_{n+m+1}) \otimes [1,1,1,0] + (\sum_{i=1}^{n+m} \mathbf{e}_i) \otimes [0,0,0,1] + D \cdot \mathbf{v}^* \in \mathbb{Z}^{(n+m+1)\times 4}$, where $D = D_{\hat{\mathcal{U}}} + n$ (and $D_{\hat{\mathcal{U}}}$, as defined above, is the degree of the polynomial computed by $\hat{\mathcal{U}}$). We note that for all $x \in \{0,1\}^n$ it holds that $\mathbf{v}_{\mathrm{zt}} = \mathbf{v}_0 + \sum_{i=1}^{n} (M[i] \cdot \mathbf{v}_{i,x_i} + \hat{\mathbf{v}}_{i,x_i}) + \sum_{i=n+1}^{n+m} M[i] \cdot \mathbf{v}_i + D \cdot \mathbf{v}^*$.

We illustrate the various level vectors in Figure 2.

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

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

$$\mathbf{v}_0 = \left[\begin{array}{ccc|c} 0 & \cdots & 0 & 1 \\ 0 & \cdots & 0 & 1 \\ 0 & \cdots & 0 & 1 \\ \hline 0 & \cdots & 0 & 0 \end{array}\right], \quad \mathbf{v}^* = \left[\begin{array}{ccc|c} 0 & \cdots & 0 & 0 \\ 0 & \cdots & 0 & 0 \\ 0 & \cdots & 0 & 0 \\ \hline 0 & \cdots & 0 & 1 \end{array}\right]$$

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

Figure 2: The level vectors for obfuscator $\mathsf{RobustObf}$.

Obfuscator $\mathsf{RobustObf}$:

- **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

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

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 $w_{i,b} = (y_i, b, \rho_{i,b,1}, \ldots, \rho_{i,b,n})$ and $y_i, \rho_{i,b,j}$ are uniform.

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 $\hat{w}_i = (\hat{y}_i, \hat{\beta}_i, \hat{\rho}_{i,1}, \ldots, \hat{\rho}_{i,n})$, 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.

As in our previous obfuscator, all of the required encodings can be efficiently generated using *params*, as explained in Remark 2.3.

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 of the ring $\mathcal{R}$.

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 $\left(R_1 = [r_1]_{\mathbf{v}_1}, Z_1 = [r_1 w_1]_{\mathbf{v}_1 + k\mathbf{v}^*}\right)$, $\left(R_2 = [r_2]_{\mathbf{v}_2}, Z_1 = [r_2 w_2]_{\mathbf{v}_2 + k\mathbf{v}^*}\right)$, operation $\mathsf{op} \in \{\times, +, -\}$.

- **Output:** Pair of encodings $\left(R^* = [r_1 r_2]_{\mathbf{v}_1+\mathbf{v}_2}, Z = [r_1 r_2 \cdot (w_1 \;\mathsf{op}\; w_2)]_{\mathbf{v}_1+\mathbf{v}_2+tk\cdot\mathbf{v}^*}\right)$, 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. 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:

$$R_{\mathcal{U}} = [r_{\mathcal{U}}]_{\mathbf{v}_{\mathcal{U}}}, \quad Z_{\mathcal{U}} = [r_{\mathcal{U}} \cdot w_{\mathcal{U}}]_{\mathbf{v}_{\mathcal{U}}+D_{\hat{\mathcal{U}}}} ,$$

   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

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

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

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

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

5.1 Useful Algebraic Tools

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

Fact 5.1. *Let $\sigma \in \mathbb{N}$, let $p_1, \ldots, p_\sigma$ be distinct primes and let $P = \prod_{i=1}^{\sigma} p_i$. Then a multivariate polynomial of total degree d has at most d^σ roots over $\mathbb{Z}_P$.*

Proof. Consider the CRT representation of root over $\mathbb{Z}_P$. Each of its component must be a root modulo the respective $\mathbb{Z}_{p_i}$. By Schwartz-Zippel the polynomial has at most d roots modulo each p_i and therefore there are at most d^σ distinct tuples where all components are roots. □

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 5.2. *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 fields, 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

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

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 5.3. *Let C be a purely arithmetic circuit (as per Definition 3.4) of size s and degree d. Then the polynomial $\|\mathcal{P}_C\|_1 \leq 2^{sd}$ (where the norm refers to the ℓ_1 norm of the coefficient vector of $\mathcal{P}_C$).*

Proof. We prove by induction. 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 $\|\mathcal{P}_C\| \leq 2^{(s-1)d_1} \cdot 2^{(s-1)d_1} \leq 2^{sd}$. 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 we get $\|\mathcal{P}_C\| \leq 2^{(s-1)d} + 2^{(s-1)d} \leq 2^{sd}$. $\square$

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

Definition 5.4. *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 $j = 1, \ldots, k$.*

Obfuscators in El-Gamal form. Recall that a canonical obfuscator is defined by length function $\ell = \ell(n)$, ring arity $\sigma(n)$, integer-valued vectors $(\mathbf{v}_1, \ldots, \mathbf{v}_\ell, \mathbf{v}_{\text{zt}})$, and a distribution $\mathcal{D}_\lambda(n, K)$ over $\ell(n)$ ring elements $(a_1, \ldots, a_\ell)$. A canonical form obfuscator is in *El-Gamal form* (EG in short) if the ring elements $(a_i)_{i \in [\ell]}$ can be partitioned to pairs $(r_i, z_i)_{\ell/2}$ where the vector $(r_1, \ldots, r_{\ell/2})$ is a vector of uniformly and independently chosen ring elements, and $z_i = r_i \cdot w_i$ for some $w_i \in \mathcal{R}$. (The same w_i may appear twice and may not be uniformly distributed, and furthermore r_i, z_i may not be encoded in the same level.)

Note that both of our obfuscators are in El-Gamal form.

5.2 Admissible Distributions on Composites and Rings

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 with the property that the min-entropy of every prime factor of $\mathcal{N}_k$ is at least $\Omega(k)$. A formal definition follows.

Definition 5.5. *An ensemble of probability distributions $\{\mathcal{N}_k\}$ is k-*admissible *if $\mathcal{N}_k$ samples a poly(k)-bit integer with the property that the min-entropy of every prime factor of $\mathcal{N}_k$ is at least $\Omega(k)$. An ensemble of probability distributions over rings $\{\mathcal{R}_k\}$ is k-admissible if $\mathcal{R}_k \cong \mathbb{Z}_N$ and the random variable N is k-admissible.*

It is not hard to see that every small fixed integer x is likely to be co-prime to $y \overset{R}{\leftarrow} \mathcal{N}_k$.

Lemma 5.6. *Let $\mathcal{N}_k$ be some k-admissible distribution. Then for all $x \in \mathbb{Z} \setminus \{0\}$, it holds that*

$$\Pr_{y \overset{R}{\leftarrow} \mathcal{N}_{t,k}} [\gcd(|x|, y) > 1] \leq \log|x| \cdot \mathit{poly}(k) \cdot 2^{-\Omega(k)} \leq \log|x| \cdot 2^{-\Omega(k)} .$$

Proof. It holds that x has at most $\log|x|$ prime factors and y has at most *poly*(k) prime factors. For every fixed i, j, the probability that the i-th prime factor of x equals to the j-th prime factor of y is at most $2^{-\Omega(k)}$. Applying the union bound, the lemma follows. □

It follows for an admissible ring distribution, any fixed (short) list of (small) integers is unlikely to hit non-invertible ring element.

Corollary 5.7. *Let $L \in \mathbb{N}$ and let $\mathcal{L} \subseteq \mathbb{Z} \setminus \{0\}$ be a list of L integers such that all $x \in \mathcal{L}$, $|x| \leq 2^{poly(\lambda)}$. Let $\mathcal{R} \cong \mathbb{Z}_N$ be a ring where N is chosen from some $(\log L + \omega(\log\lambda))$-admissible distribution. Then, the probability that there exists $x \in \mathcal{L}$ which is not a unit in $\mathcal{R}$ is negl(λ).*

5.3 Proof Outline

In this section we describe the common general outline of the proof that will be applied both to SimpleObf (Section 5.4) and to RobustObf (Section 5.5).

Since our constructions are in canonical form (in fact, in El-Gamal form) it suffices, by Lemma 3.7, to prove algebraic security according to Definition 3.6.

Fix some function identifier K and polynomial P. We note that P is associated with a purely arithmetic circuit of polynomial size and polynomial degree. The latter is since the degree cannot go above $\|\mathbf{v}_{\mathrm{zt}}\|_1$. Since our obfuscator is in EG form, we can re-write P as a sum of terms of the form

$$M(r) \cdot Q(w) ,$$

where M is a monomial and Q is a polynomial. It suffices to show that given an oracle access to C_K, we can determine if the above product equals to zero with more than negligible probability (where the probability is taken over $\mathcal{D}_\lambda(K)$).

In the simulation we use the min-entropy of the orders p_i of the sub-rings of $\mathcal{R}$ as follows. We present a simulator that needs not know any information about the order of $\mathcal{R}$ or its sub-rings. However, this simulator succeeds only as long as a list of non-zero integers $\mathcal{L}$ generated during the simulation does not contain any element that is not a unit in $\mathcal{R}$. The length of the list will be bounded by $2^{\|\mathbf{v}_{\mathrm{zt}}\|_1} \cdot \mathit{poly}(\lambda)$, and the absolute value of each of these numbers will be at most $2^{poly(\lambda)}$. Fact 5.3 and Corollary 5.7 thus guarantee that the simulation fails only with negligible probability.

Formally, our simulator (which is oblivious to the order of the ring) is going to have the following properties:

1. The simulator will generate, as a by product, a list $\mathcal{L}$ of $L = 2^{\|\mathbf{v}_{\mathrm{zt}}\|_1} \cdot poly(\lambda)$ integers of absolute value at most $2^{poly(\lambda)}$. In particular, the list is a subset of the coefficients of the polynomial P. Since P is computable by a purely arithmetic circuit of size $poly(\lambda)$ and degree at most $\|\mathbf{v}_{\mathrm{zt}}\|_1$, the bounds will follow from Fact 5.3.

2. We will prove that as long as all of the elements of $\mathcal{L}$ (cast into $\mathcal{R}$) are units in $\mathcal{R}$, the simulation is successful.

3. The distribution of $\mathcal{R}$ as described in our model will guarantee, by Corollary 5.7, that the event of simulation failure due to $\mathcal{L}$ containing a non-unit is negligible.

The specifics of applying this outline will vary between the specific obfuscators.

5.4 Algebraic Security Proof for SimpleObf

Theorem 5.8. *The obfuscator* SimpleObf *is secure relative to the GES oracle $\mathcal{MRG}$ defined over any* $(\|\mathbf{v}_{zt}\|_1 + \omega(\log(\lambda)))$-admissible *ring distribution.*

We follow the outline from Section 5.3. We start with structural claims on Q, viewed as a polynomial over the integers.

Lemma 5.9. *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. Assume towards contradiction that there exists an i such that Q is neither $w_{i,0}$-free nor $w_{i,1}$-free. This implies that $r_{i,0} \cdot r_{i,1} | M(r)$, and therefore that $M(r)/(r_{i,0} \cdot r_{i,1})$ has level $\mathbf{v}_{\mathrm{zt}} - (\mathbf{v}_{i,0} + \mathbf{v}_{i,1})$. However, this level is not reachable by any combination of elements provided by the obfuscator (see Figure 1). Contradiction follows. □

Lemma 5.10. *The polynomial Q is $\left\{w_{i,b}^{(M[i]+1)}, w_i^{(M[i]+1)}, \hat{w}_i^2, w_0^2\right\}_{i\in[n], b\in\{0,1\}}$-free.*

Proof. This follows immediately by considering the levels associated with the monomial mentioned in the proof. All of these elements can only exist at levels $\mathbf{v} \not\leq \mathbf{v}_{\mathrm{zt}}$ and therefore their respective r values cannot divide M. The lemma follows. □

Lemma 5.11. *The polynomial $P(r, w)$ can be written as a sum of at most $T = 2^{\|\mathbf{v}_{zt}\|_1}$ terms of the form $M(r)Q(w)$, where M is a monomial.*

Proof. We count how many possible monomials of the form $M(r)$ can exist in P. The previous lemmas assert that $M(r)$ is "associated" with a string $(x_1, \dots, x_n)$, such that $r_{i,1-x_i} \not| M(r)$, and the multiplicity of r_{i,x_i} is at most $M[i]$. Similarly, for $i = n+1, \dots, n+m$, it holds that r_i has degree at most $M[i]$. Lastly, $\hat{r}_{i,b}$ has degree at most 1, and likewise r_0. A straightforward combinatorial argument shows that

$$T \leq 2^n \cdot (\prod_{i\in[n+m]} (M[i] + 1)) \cdot 2^{2n+1} \leq 2^{3n+1+\sum_{i\in[n+m]} M[i]} \leq 2^{\|\mathbf{v}_{\mathrm{zt}}\|_1} .$$

□

We can now distinguish between two cases.

1. It holds that

$$\left(w_0 - (\prod_{i\in[n]} \hat{w}_i) \cdot \hat{\mathcal{U}}(w_{1,x_1}, \ldots, w_{n,x_n}, w_{n+1}, \ldots, w_{n+m})\right) \not| Q$$

for any $x = (x_1, \ldots, x_n) \in \{0,1\}^n$.

2. There exists $x = (x_1, \ldots, x_n) \in \{0,1\}^n$ such that

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

Lemma 5.12 (Case 1)**.** *Let $x = (x_1, \ldots, x_n) \in \{0,1\}^n$ be the value guaranteed to exist in Lemma 5.9. If it holds that*

$$\left(w_0 - (\prod_{i\in[n]} \hat{w}_i) \cdot \hat{\mathcal{U}}(w_{1,x_1}, \ldots, w_{n,x_n}, w_{n+1}, \ldots, w_{n+m})\right) \not| Q \ , \tag{2}$$

then

$$\Pr_{\mathcal{D}_\lambda(K)}[Q = 0] = negl(\lambda) \ .$$

Proof. We recall that Q is $\{w_{i,1-x_i}\}_{i\in[n]}$-free by Lemma 5.9, and therefore there exists a polynomial Q' such that

$$Q(w) = Q'(w_0, \hat{w}_1, \ldots, \hat{w}_n, w_{1,x_1}, \ldots, w_{n,x_n}, w_{n+1}, \ldots, w_{n+m}) \ .$$

We further define the polynomial $Q''(\hat{w}_1, \ldots, \hat{w}_n, w_{1,x_1}, \ldots, w_{n,x_n}, w_{n+1}, \ldots, w_{n+m})$ as the polynomial obtained from Q' by assigning $w_0 = \left(\prod_{i\in[n]} \hat{w}_i\right) \cdot \hat{\mathcal{U}}(w_{1,x_1}, \ldots, w_{n,x_n}, w_{n+1}, \ldots, w_{n+m})$. Since $w_0 - \left(\prod_{i\in[n]} \hat{w}_i\right) \cdot \hat{\mathcal{U}}(w_{1,x_1}, \ldots, w_{n,x_n}, w_{n+1}, \ldots, w_{n+m})$ does not divide Q (Eq. (2)), the polynomial $Q''(w)$ is not identically zero.[12] Therefore, Q'' contains at least one non-zero monomial. Let a be the coefficient of this monomial, and inset a into the list $\mathcal{L}$.

Now consider $Q''(w)[\![1]\!] = Q''[\![1]\!](w[\![1]\!])$. Since $w[\![1]\!]$ is the set of y variables which are uniform and independent, and since $Q''[\![1]\!]$ contains at least one non-zero monomial (by the assumption that a is a unit in $\mathcal{R}$ and therefore also in the first sub-ring), it follows that

$$\begin{aligned}\Pr_{\mathcal{D}_\lambda(K)}[Q(w) = 0] &= \Pr_{\mathcal{D}_\lambda(K)}[Q''(w) = 0] \\ &\le \Pr_{\mathcal{D}_\lambda(K)}[Q''(w)[\![1]\!] = 0] \\ &= \Pr_{\mathcal{D}_\lambda(K)}[Q''[\![1]\!](w[\![1]\!]) = 0] = negl(\lambda) \ .\end{aligned}$$

□

Lemma 5.13. *Let $x = (x_1, \ldots, x_n) \in \{0,1\}^n$ be the value guaranteed to exist in Lemma 5.9. If it holds that*

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

[12] This implication is standard when the polynomials are defined over a field, but it also holds for integer polynomials. See Fact 5.2.

then there exists a constant a' such that

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

Proof. By Lemma 5.10 and Lemma 5.9, the degrees of all variables in $Q(w)$ cannot be higher than their degrees in $(w_0 - \big(\prod_{i \in [n]} \hat{w}_i \big) \cdot \hat{\mathcal{U}}(w_{1,x_1}, \ldots, w_{n,x_n}, w_{n+1}, \ldots, w_{n+m}))$. The lemma follows. $\square$

Lemma 5.14 (Case 2)**.** *If there exists $x = (x_1, \ldots, x_n) \in \{0,1\}^n$ such that*

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

Then if $C_K(x) = 0$ then $\Pr_{\mathcal{D}_\lambda(K)}[Q = 0] = negl(\lambda)$, and if $C_K(x) = 1$ then $\Pr_{\mathcal{D}_\lambda(K)}[Q = 0] = 1$.

Proof. Let x, Q be as in the lemma statement, and insert a into the list $\mathcal{L}$ (we therefore assume from this point and on that it is a unit in $\mathcal{R}$). We first note that for all $i \neq 2$, it holds that $\Pr_{\mathcal{D}_\lambda(K)}[Q[\![i]\!] = 0] = 1$. Namely, Q is the zero polynomial over all sub-rings except the second. This follows immediately by definition.

Let us thus examine $Q[\![2]\!](w) = Q(w[\![2]\!])$. We have that

$$\begin{aligned} \Pr_{\mathcal{D}_\lambda(K)}[Q = 0] &= \Pr_{\mathcal{D}_\lambda(K)}[Q[\![2]\!] = 0] \\ &= \Pr_{\mathcal{D}_\lambda(K)}[a[\![2]\!] \cdot \prod_{i \in [n]} \hat{\beta}_i \cdot (1 - \hat{\mathcal{U}}(x_1, \ldots, x_n, K_1, \ldots, K_m)) = 0] \\ &= \Pr_{\mathcal{D}_\lambda(K)}[a[\![2]\!] \cdot \prod_{i \in [n]} \hat{\beta}_i \cdot (1 - C_K(x)) = 0] . \end{aligned}$$

Since $a[\![2]\!] \neq 0$, it follows that if $C_K(x) = 0$ then this equals to $\Pr_{\mathcal{D}_\lambda(K)}[a \cdot \prod_{i \in [n]} \hat{\beta}_i = 0] = negl(\lambda)$, whereas if $C_K(x) = 1$ then this is $\Pr_{\mathcal{D}_\lambda(K)}[0 = 0] = 1$. $\square$

Overall, the simulator can determine whether Q evaluates to zero. In the first case, it will simply say that Q does not evaluates to zero, and in the second case it will test if $C_K(x) = 1$ and, only if this test passes, it will output Yes (meaning that Q evaluates to zero). By lemmas 5.12 and 5.14 the simulator errs with no more than negligible probability.

5.5 Algebraic Security Proof for RobustObf

We will prove the following theorem.

Theorem 5.15. *The obfuscator* RobustObf *is secure relative to the GES oracle $\mathcal{URG}$ defined over any* $(\|\mathbf{v}_{zt}\|_1 + \omega(\log(\lambda)))$-admissible *ring distribution.*

Since we aim for security relative to $\mathcal{MRG}$ (using Lemma 3.7), we should take into account the possibility that the adversary P is of level smaller than $\mathbf{v}_{\text{zt}}$. We follow the outline from Section 5.3 by viewing P as a sum of terms of the form $M(r) \cdot Q(w)$. We will analyze the probability that $Q(\mathcal{D}_\lambda(K))$ evaluates to zero, starting with a few structural claims on Q.

Lemma 5.16. *There exists a constant a and a w_0-free polynomial $Q'(w)$ such that*

$$Q(w) = a \cdot w_0 - Q'(w) \ .$$

Proof. First of all, we note that the structure of our sets prevents w_0 from being multiplied by any of the other w variable. This is because w_0 is encoded at level $\geq (D+n)\mathbf{v}^*$, and the other w variables are encoded at level $\geq \mathbf{v}^*$. It follows that any product of w_0 and another w variable will be at level $\geq (D+n+1)\mathbf{v}^*$. However, since $(D+n+1) \not\leq \mathbf{v}_{\mathrm{zt}}$, contradiction follows. □

Lemma 5.17. *For all $i \in [n]$, the polynomial Q (and therefore also Q' from Lemma 5.16) is $\hat{w}_i^2$-free.*

Proof. Assume towards contradiction that this is not the case. Then it means that $\hat{r}_{i,b_1} \cdot \hat{r}_{i,b_2} | M$ for some $b_1, b_2 \in \{0,1\}$ (because the w variables cannot separate from their companion r values). This means that the level of P is at least $\hat{\mathbf{v}}_{i,b_1} + \hat{\mathbf{v}}_{i,b_2}$. However, for any values of b_1, b_2 it holds that $\hat{\mathbf{v}}_{i,b_1} + \hat{\mathbf{v}}_{i,b_2} \not\leq \mathbf{v}_{\mathrm{zt}}$ and contradiction follows. □

We can now distinguish between three main cases.

1. It holds that $\left(\prod_{i\in[n]} \hat{w}_i\right) \not| Q'(w)$.

2. It holds that $\left(\prod_{i\in[n]} \hat{w}_i\right) | Q'(w)$, namely there exists a polynomial $Q''(w)$ which is $\{w_0, \hat{w}_1, \ldots, \hat{w}_n\}$-free and
$$Q(w) = aw_0 - \big(\prod_{i\in[n]} \hat{w}_i\big) \cdot Q''(w) \ .$$
However,
$$Q''(w) \neq a \cdot \hat{\mathcal{U}}(w_{1,x_1}, \ldots, w_{n,x_n}, w_{n+1}, \ldots, w_{n+m})$$
for any $x = (x_1, \ldots, x_n) \in \{0,1\}^n$.

3. There exists $x = (x_1, \ldots, x_n) \in \{0,1\}^n$ such that
$$Q(w) = a \cdot \left(w_0 - \left(\prod_{i\in[n]} \hat{w}_i\right) \cdot \hat{\mathcal{U}}(w_{1,x_1}, \ldots, w_{n,x_n}, w_{n+1}, \ldots, w_{n+m}) \right) \ . \qquad (3)$$

Lemma 5.18 (Case 1)**.** *If $\exists i.\ \hat{w}_i \not| Q'(w)$, then*

$$\Pr_{w \xleftarrow{R} \mathcal{D}_\lambda(K)} [Q(w) = 0] = \mathit{negl}(\lambda) \ .$$

Proof. Assume towards contradiction that there exists such i. Consider the evaluation of Q over the ith sub-ring, $Q(w)[\![i]\!]$. Since $w_0[\![i]\!]_{|\mathcal{D}_\lambda(K)} = 0$ for all i, it holds that $Q(w)[\![i]\!]_{|\mathcal{D}_\lambda(K)} = Q'(w)[\![i]\!]_{|\mathcal{D}_\lambda(K)}$. Recall that $Q'(w)[\![i]\!] = Q'[\![i]\!](w[\![i]\!])$, which is exactly the evaluation of $Q'[\![i]\!]$ on the respective $\rho_{*,i}$ (= the variables $\rho_{j,b,i}, \rho_{j,i}, \hat{\rho}_{j,i}$) values which are all uniform and independent in the ith sub-ring. Since Q' is not identically zero, letting a be one of its non-zero coefficients, and inserting a into the list $\mathcal{L}$, guarantees that $Q'[\![i]\!]$ is also not identically zero.

Fact 5.1 now guarantees that $\Pr_{w \xleftarrow{R} \mathcal{D}_\lambda(K)}[Q'[\![i]\!](\rho_{*,i}) = 0] = \mathit{negl}(\lambda)$ and therefore that $\Pr_{w \xleftarrow{R} \mathcal{D}_\lambda(K)}[Q(w) = 0] = \mathit{negl}(\lambda)$. □

Lemma 5.19. *If $Q'(w) = \left(\prod_{i\in[n]} \hat{w}_i\right)\cdot Q''$ for some Q'', then there exists $x = (x_1,\ldots,x_n) \in \{0,1\}^n$ such that Q'' is $\{w_{i,1-x_i}\}_{i\in[n]}$-free.*

We recall that by Lemma 5.17, $Q''(w)$ is also $\{\hat{w}_i\}_{i\in[n]}$-free.

Proof. Assume towards contradiction that this is not the case. Namely that there exists $i \in [n]$ such that $Q''(w)$ is neither $w_{i,0}$-free nor $w_{i,1}$-free. This means that $r_{i,0}\cdot r_{i,1}|M(r)$. However it also holds that $\hat{r}_{i,b}|M(r)$ for some $b \in \{0,1\}$. The latter is since $\hat{w}_i|Q'$ and thus Q is not $\hat{w}_i$-free. However, for all $b\in\{0,1\}$, $\mathbf{v}_{i,0}+\mathbf{v}_{i,1}+\hat{\mathbf{v}}_{i,b}\not\leq \mathbf{v}_{\mathrm{zt}}$ and contradiction follows. □

Lemma 5.20 (Case 2)**.** *If $Q' = \left(\prod_{i\in[n]} \hat{w}_i\right)\cdot Q''$ and for the x from Lemma 5.19 it holds that $Q'' \neq a\cdot\hat{\mathcal{U}}(w_{1,x_1},\ldots,w_{n,x_n},w_{n+1},\ldots,w_{n+m})$, then $\Pr_{\mathcal{D}_\lambda(K)}[Q=0] = negl(\lambda)$.*

Proof. Consider a non-zero monomial of the polynomial $a\cdot\hat{\mathcal{U}}(w)-Q''(w)$, and let a' be its coefficient. Add a' to the list $\mathcal{L}$ so we can assume from now on that it is a unit in $\mathcal{R}$. This implies that $\left(a\cdot\hat{\mathcal{U}}(\cdot)-Q''(\cdot)\right)[\![1]\!] \not\equiv 0$.

We will consider the evaluation of Q on the first sub-ring, $Q[\![1]\!]$. It holds that $w_0[\![1]\!] = \prod_{i\in[n]}\hat{y}_i\cdot y_0$, and that $Q'[\![1]\!] = \prod_{i\in[n]}\hat{y}_i\cdot Q''[\![1]\!]$. Therefore, recalling that $\hat{y}_i$ are uniform and independent,

$$\begin{aligned}\Pr_{\mathcal{D}_\lambda(K)}[Q(w)=0] &\leq \Pr_{\mathcal{D}_\lambda(K)}[Q[\![1]\!](w[\![1]\!])=0]\\ &\leq \Pr_{\mathcal{D}_\lambda(K)}[\prod_{i\in[n]}\hat{y}_i=0] + \Pr_{\mathcal{D}_\lambda(K)}[a[\![1]\!]y_0 - Q''(y)[\![1]\!]=0]\\ &\leq \Pr_{\mathcal{D}_\lambda(K)}[a[\![1]\!]y_0 - Q''(y)[\![1]\!]=0] + negl(\lambda)\ .\end{aligned}$$

By Lemma 5.19, it holds that there exists a polynomial $\hat{Q}$ such that

$$Q''(w) = \hat{Q}(w_{1,x_1},\ldots,w_{n,x_n},w_{n+1},\ldots,w_{n+m})\ .$$

It follows that

$$\begin{aligned}\Pr_{\mathcal{D}_\lambda(K)}[ay_0 - Q''(y)[\![1]\!]=0] &= \Pr_{\mathcal{D}_\lambda(K)}[ay_0 - \hat{Q}(y_1,\ldots,y_n,y_{n+1},\ldots,y_{n+m})=0]\\ &= \Pr_{\mathcal{D}_\lambda(K)}[a\cdot\hat{\mathcal{U}}(y_1,\ldots,y_n,y_{n+1},\ldots,y_{n+m}) - \hat{Q}(y_1,\ldots,y_n,y_{n+1},\ldots,y_{n+m})=0]\ .\end{aligned}$$

However, as we showed above, $\left(a\cdot\hat{\mathcal{U}}(\cdot)-Q''(\cdot)\right)[\![1]\!]\not\equiv 0$, and therefore the probability of zero is negligible here. □

Lemma 5.21 (Case 3)**.** *If there exists $x=(x_1,\ldots,x_n)\in\{0,1\}^n$ such that*

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

Then if $C_K(x)=0$ then $\Pr_{\mathcal{D}_\lambda(K)}[Q=0]=negl(\lambda)$, and if $C_K(x)=1$ then $\Pr_{\mathcal{D}_\lambda(K)}[Q=0]=1$.

Proof. Let x, Q be as in the lemma statement. We first note that for all $i \neq 2$, it holds that $\Pr_{\mathcal{D}_\lambda(K)}[Q[\![i]\!] = 0] = 1$. Namely, Q is the zero polynomial over all sub-rings except the second. This follows immediately by definition.

Let us thus examine $Q(w)[\![2]\!] = Q[\![2]\!](w[\![2]\!])$. We have that

$$\begin{aligned}\Pr_{\mathcal{D}_\lambda(K)}[Q(w) = 0] &= \Pr_{\mathcal{D}_\lambda(K)}[Q(w)[\![2]\!] = 0] \\ &= \Pr_{\mathcal{D}_\lambda(K)}[a[\![2]\!] \cdot \prod_{i\in[n]} \hat{\beta}_i \cdot (1 - \hat{\mathcal{U}}(x_1, \ldots, x_n, K_1, \ldots, K_m)) = 0] \\ &= \Pr_{\mathcal{D}_\lambda(K)}[a[\![2]\!] \cdot \prod_{i\in[n]} \hat{\beta}_i \cdot (1 - C_K(x)) = 0] \ .\end{aligned}$$

We will place a in the list $\mathcal{L}$, which will allow us to assume that $a[\![2]\!]$ is a unit in the second sub-ring. It follows that if $C_K(x) = 0$ then this equals to $\Pr_{\mathcal{D}_\lambda(K)}[a[\![2]\!] \cdot \prod_{i\in[n]} \hat{\beta}_i = 0] = \mathit{negl}(\lambda)$, whereas if $C_K(x) = 1$ then this is $\Pr_{\mathcal{D}_\lambda(K)}[0 = 0] = 1$. □

Overall, the simulator can determine whether Q evaluates to zero. In the first and second cases, it will simply say that Q does not evaluates to zero, and in the third case it will test if $C_K(x) = 1$ and, only if this test passes, it will output Yes (meaning that Q evaluates to zero). By lemmas 5.18, 5.20 and 5.21 the simulator errs with no more than negligible probability.

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

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.

[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.

[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.

[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.

[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. `http://eprint.iacr.org/`.

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