Virtual Black-Box Obfuscation for All Circuits via Generic Graded Encoding

Zvika Brakerski[1] and Guy N. Rothblum[2]

[1] Weizmann Institute of Science
[2] Microsoft Research

Abstract. We present a new general-purpose obfuscator for all polynomial size circuits. The obfuscator uses graded encoding schemes, a generalization of multilinear maps. We prove that the obfuscator exposes no more information than the program's black-box functionality, and achieves *virtual black-box security*, in the generic graded encoded scheme model. This proof is under the Bounded Speedup Hypothesis (BSH, a plausible worst-case complexity-theoretic assumption related to the Exponential Time Hypothesis), in addition to standard cryptographic assumptions. We also prove that it satisfies the notion of *indistinguishability obfuscation* without without relying on BSH (in the same generic model and under standard cryptographic assumptions).

Very recently, Garg et al. (FOCS 2013) used graded encoding schemes to present a candidate obfuscator for indistinguishability obfuscation. They posed the problem of constructing a provably secure indistinguishability obfuscator in the generic graded encoding scheme model. Our obfuscator resolves this problem (indeed, under BSH it achieves the stronger notion of virtual black box security, which is our focus in this work).

Our construction is different from that of Garg et al., but is inspired by it, in particular by their use of permutation branching programs. We obtain our obfuscator by developing techniques used to obfuscate d-CNF formulas (ITCS 2014), and applying them to permutation branching programs. This yields an obfuscator for the complexity class $\mathcal{NC}^1$. We then use homomorphic encryption to obtain an obfuscator for any polynomial-size circuit.

1 Introduction

Code obfuscation is the task of taking a program, and making it "unintelligible" or impossible to reverse engineer, while maintaining its input-output functionality. While this is a foundational question in the theory and practice of cryptography, until recently very few techniques or heuristics were known. Recently, however, several works have leveraged new constructions of cryptographically secure graded encoding schemes (which generalize multilinear maps) [23, 20] to propose obfuscators for complex functionalities [10, 11] and, in a fascinating recent work of Garg et al [24], even for arbitrary polynomial size circuits.

In this work, we propose a new code obfuscator, building on techniques introduced in [10, 24, 11]. The obfuscator works for any polynomial-time circuit, and its security is analyzed in the idealized generic graded encoding scheme model. We prove that, in this idealized model, the obfuscator achieves the strong "virtual black-box" security notion of Barak et al. [5] (see below). Security in the idealized model relies on a worst-case exponential assumption on the hardness of the $\mathcal{NP}$-complete 3SAT problem (in the flavor of the well known exponential time hypothesis). Our construction relies on *asymmetric graded encoding schemes*, and can be instantiated using the new candidate constructions of Garg, Gentry and Halevi [23], or of Coron, Lepoint and Tibouchi [20].

Obfuscation: Definitions. Intuitively, an obfuscator should generate a new program that preserves the the original program's functionality, but is impossible to reverse engineer. The theoretical study of this problem was initiated by Barak et al. [5]. They formalized a strong simulation-based security requirement of *black box obfuscation*: namely, the obfuscated program should expose nothing more than what can be learned via oracle access to its input-output behavior. We refer to this notion as "black-box" obfuscation, and we use this strong formalization throughout this work.

A weaker notion of obfuscation, known as *indistinguishability* or *best-possible* obfuscation was studied in [5, 27]. An indistinguishability obfuscator guarantees that the obfuscations of any two programs (boolean circuits) with identical functionalities are indistinguishable. We note that, unlike the black-box definition of security, indistinguishability obfuscation does not quantify or qualify what information the obfuscation might expose. In particular, the obfuscation might reveal non-black-box information about the functionality. Recently, Sahai and Waters [37] showed that indistinguishability obfuscation suffices for many cryptographic applications, such as transforming private key cryptosystems to public key, and even for constructing deniable encryption schemes.

Prior Work: Negative Results. In their work, [5] proved the *impossibility* of general-purpose black-box obfuscators (i.e. ones that work for any polynomial-time functionality) in the virtual black box model. This impossibility result was extended by Goldwasser and Kalai [26]. Goldwasser and Rothblum [27] showed obstacles to the possibility of achieving indistinguishability obfuscation with information-theoretic security, and to achieving it in the idealized random oracle model.

Looking ahead, we note that the impossibility results of [5, 26] *do not extend* to idealized models, such as the random oracle model, the generic group model, and (particularly relevant to our work) the generic graded encoding model.

Prior Work: Positive Results. Positive results on obfuscation focused on specific, simple programs. One program family, which has received extensive attention, is that of "point functions": password checking programs that only accept a single input string, and reject all others. Starting with the work of Canetti [13], several works have shown obfuscators for this family under various assumptions [17, 32, 39], as well as extensions [14, 8]. Canetti, Rothblum and Varia [18] showed how to obfuscate a function that checks membership in a hyperplane of constant dimension (over a large finite field). Other works showed how to obfuscate cryptographic function classes under different definitions and formalizations. These function classes include checking proximity to a hidden point [21], vote mixing [1], and re-encryption [29]. Several works [13, 17, 28, 29] relaxed the security requirement so that obfuscation only holds for a random choice of a program from the family.

More recently, Brakerski and Rothblum [10] showed that graded encoding schemes could be used to obfuscate richer function families. They constructed a black-box obfuscator for *conjunctions*, the family of functions that test whether a subset of the input bits take on specified values. Building on this result, in a followup work [11], they constructed a black-box obfuscator for d-CNFs and (more generally) conjunctions of $\mathcal{NC}^0$ circuits. These constructions were proved secure in the generic graded encoding model. The conjunction obfuscator was also shown to be secure under falsifiable (see [34]) multilinear DDH-like assumptions, so long as the conjunction is drawn from a family with sufficient entropy.

In recent work, Garg et al. [24] use cryptographic graded encoding schemes to construct a candidate indistinguishability obfuscator (see above) for all polynomial size circuits. This is the first non-trivial candidate in the literature for general-purpose obfuscation. The main differences between our results and theirs are: (i) we construct an obfuscator with the stronger security notion of black-box obfuscation (for the same class of functions), and (ii) we provide a security proof in the generic graded encoding scheme model. This was posed as a major open question in [24].[3]

Canetti and Vaikuntanathan [19] outline a candidate obfuscator and prove its security in an idealized pseudo-free group model. They also use Barrington's theorem and randomization techniques. The main difference from our work is in the nature of their idealized pseudo-free group model: in particular, we do not know of an instantiation that is conjectured to be secure.

1.1 Our Work: Black-Box Obfuscation for all of $\mathcal{P}$

In this work we construct an obfuscator for any function in $\mathcal{P}$, using cryptographic graded encoding schemes. Our obfuscator can be instantiated using recently proposed candidates [23, 20]. The main component of our construction is an obfuscator for the complexity class $\mathcal{NC}^1$, which is then leveraged to an obfuscator for $\mathcal{P}$ using homomorphic encryption. Our main contribution is a proof that the main component is a secure black-box obfuscator in the generic graded encoding scheme model, assuming the *bounded speedup hypothesis* (BSH) [11], a generalization of the exponential time hypothesis. More details follow.

Theorem 1.1. *There exists an obfuscator* PObf *for any circuit in* $\mathcal{P}$*, which is virtual black-box secure in the generic graded encoding scheme model, assuming the bounded speedup hypothesis, and the existence of homomorphic encryption with an* $\mathcal{NC}^1$ *decryption circuit.*

[3] [24] provide a proof of security in a more restricted "generic colored matrix model".

We also prove that our obfuscator is an indistinguishability obfuscator in the generic graded encoding scheme model (in fact, we show that this is true even for a simplified variant of the construction). This proof does not require the bounded speedup hypothesis.[4]

Theorem 1.2. *There exists an obfuscator* $\mathsf{PIndObf}$ *for any circuit in* $\mathcal{P}$*, which is an indistinguishability obfuscator in the generic graded encoding scheme model, assuming the existence of homomorphic encryption with an* $\mathcal{NC}^1$ *decryption circuit.*

To prove Theorem 1.2, we use an equivalent formulation for the security of indistinguishability obfuscators. This formulation requires the existence of a *computationally unbounded* simulator, which only has *black-box access* to the obfuscated program.

For the remainder of this section, we focus our attention on black-box obfuscation. Our construction proceeds in two steps. As hinted above, the first (and main) step is an obfuscator for $\mathcal{NC}^1$ circuits. Then, in the second step, we use homomorphic encryption [35, 25] to obfuscate any polynomial-size circuit. (Which is done by encrypting the input circuit using the homomorphic scheme, and obfuscating a "verified decryption" circuit, as explained in the full version [12].)

Our obfuscator for $\mathcal{NC}^1$ circuits combines ideas from: (*i*) the *d*-CNF obfuscator of [11] and its security proof. In particular, we build on their technique of *randomizing sub-assignments* to prove security in the generic model based on the bounded speedup hypothesis, and we also build on their use of random generators in each ring of the graded encoding scheme. We also use ideas from (*ii*) the indistinguishability obfuscator of [24], in particular their use of Barrington's theorem [6] and randomization techniques for permutation branching programs. We note that the obfuscator of [19] was also based on Barrington's theorem and randomization techniques. See Section 1.2 for an overview of the construction and its proof, Section 3 for the detailed construction, and the full version [12] for the security proof.

The Generic Graded Encoding Scheme Model. We prove that our construction is a black-box obfuscator in the generic graded encoding scheme model. In this model, an adversary must operate independently of group elements' representations. The adversary is given arbitrary strings representing these elements, and can only manipulate them using oracles for addition, subtraction, multilinear operations and more. See Section 2.5 for more details.

The Bounded Speedup Hypothesis. We prove security based on the *Bounded Speedup Hypothesis*, as introduced by [11]. This is a worst-case assumption about exponential hardness of 3SAT, a strengthening of the long-standing exponential time hypothesis (ETH) for solving 3SAT [30]. The exponential-time hypothesis states that no sub-exponential time algorithm can resolve satisfiability of 3CNF

[4] Under the BSH, the claim follows immediately from Theorem 1.1, because any virtual black-box obfuscator is also an indistinguishability obfuscator [27].

formulas. Intuitively, the bounded-speedup hypothesis states that no polynomial-time algorithm for resolving satisfiability of 3CNFs can have "super-polynomial speedup" over a brute-force algorithm that tests assignments one-by-one. More formally, there does not exist an ensemble of polynomial-size circuits $\{\mathcal{A}_n\}$, and an ensemble of super-polynomial-size sets of assignments $\{\mathcal{X}_n\}$, such that on input a 3CNF Φ on n-bit inputs, w.h.p. $\mathcal{A}_n$ finds a satisfying assignment for Φ in $\mathcal{X}_n$ if such an assignment exists. We emphasize that this is a worst-case hypothesis, i.e. it only says that for every ensemble $\mathcal{A}$, there *exists* some 3CNF on which $\mathcal{A}$ fails. See Section 2.2 for the formal definition.[5]

Perspective. Barak et al. [5] show that there are function families that are impossible to obfuscate under the black-box security definition. Their results *do not apply to idealized models* such as the random oracle model, the generic group model, and the generic graded encoding model. This is because their adversary needs to be able to execute the obfuscated circuit on parts of its own explicit description. In idealized models, the obfuscated circuit does not have a succinct explicit description, and so these attacks fail. Indeed, our main result, Theorem 1.1, shows that *general-purpose black-box obfuscation is possible in the generic graded encoding model* (under plausible assumptions). Indeed, prior works have shown that virtual black-obfuscation is possible in various idealized models: [5] showed that there *exists* an (arguably contrived) oracle that allows general-purpose obfuscation. [19] proposed a black-box obfuscator in an idealized generic pseudo-free group model, but we do not know an instantiation of this model that is conjectured to be secure. In contrast, Theorem 1.1 provides an (arguably) *natural* idealized model that allows general-purpose black-box obfuscation. It is natural to ask how one should interpret this result in light of the impossibility theorems.

One immediate answer, is that if one implements a graded encoding scheme using opaque secure hardware (essentially implementing the generic model), then the hardware can be used to protect any functionality (under plausible assumptions). The hardware is (arguably) natural, simple, stateless, and independent of the functionality being obfuscated.

Another answer, is that the security proof shows that (under plausible assumptions) our obfuscator is provably resilient to attacks from a rich family: namely, to all attacks that are independent of the encoding scheme's instantiation. While we find this guarantee to be of value, we caution that it should not be over-interpreted. The results of [5] imply that, for any concrete instantiation of the graded encoding scheme, the obfuscation of their unobfuscatable functions *is not secure.* In particular, their result (applied to our construction) provides a *non-generic attack* against any graded encoding candidate. This is similar to the result of [15], showing an attack against any instantiation of a random oracle in a particular protocol. Somewhat differently from their result, however, in our

[5] We note that if both the adversary and the black-box simulator are allowed to run in quasi-polynomial time, security can be based on the (standard) Exponential-Time Hypothesis.

case the primitive in question is altogether impossible in the standard model. In the result of [15], the primitive is not achieved by a specific construction when the idealized model is instantiated with *any* concrete functionality. We find this state of affairs to be of interest, even irrespective of the applications to code obfuscation.

Taking a more optimistic view, the new construction invites us to revisit limits in the negative results: both in the unobfuscatable functionalities, and in the nature of the attacks themselves. It may suggest new relaxations that make obfuscation achievable in standard models, e.g. obfuscating functionalities that inherently do not allow self-execution, or protecting against a class of attackers that cannot execute the obfuscated code on itself.

Finally, the relaxed notion of indistinguishability obfuscation, where only limited hardness and impossibility results are known, remains a promising avenue for future research. Our construction is the first provably secure indistinguishability obfuscator in the generic graded encoding model. It is interesting to explore whether indistinguishability obfuscation can be proved in the standard model under falsifiable assumptions.

Follow-up Work. In recent follow-up work, Barak et al. [4] propose an obfuscator that achieves virtual black-box security in the generic graded encoding scheme model without relying on the BSH. Their construction builds on encoding and randomization techniques introduced in this work.

1.2 Construction Overview

We proceed with an overview of the main step in our construction: an obfuscator for $\mathcal{NC}^1$. We are assuming basic familiarity with graded encoding schemes. The full construction appears in Section 3. Due to space limitations, the proof of its security in the generic model appears in the full version of this manuscript [12].

Permutation Branching Programs. The obfuscator $\mathsf{NC^1Obf}$ takes as input an $\mathcal{NC}^1$ program, represented as an oblivious width 5 permutation branching program C, as in [24]. Let m denote the depth of C (as is necessary, we allow the obfuscator to expose m or some upper bound thereof). Let $C = \{M_{j,0}, M_{j,1}\}_{j \in [m]}$, where $M_{j,b} \in \{0,1\}^{5 \times 5}$ are matrices, and let $i = \ell(j)$ indicate which variable x_i controls the jth level branch. See Section 2.1 for more background on (oblivious) branching programs.

Graded Encoding Schemes. We begin by recalling the notion of multilinear maps, due to Boneh and Silverberg [9]. Rothblum [36] considered the asymmetric case, where the groups may be different. The obfuscator makes (extensive) use of an asymmetric graded encoding schemes, which are an extension of multilinear maps.

Similarly to [24, 19], we assign a group prog_j to each level j of the branching program, and encode the matrices $M_{j,b}$ in the group prog_j.[6] This encoding is

[6] In the setting of multilinear maps, we typically refer to input groups. The map takes a single element from each input group, and maps them to a target group. In the

done as in [10, 11]: we encode each matrix $M_{j,b}$ relative to a unique generator of prog_j (denoted $\rho_{\mathsf{prog}_j,b}$), and also provide an encoding of the generators. In other words, in the j-th group prog_j, we have two pairs:

$$(\rho_{\mathsf{prog}_j,0}, (\rho_{\mathsf{prog}_j,0} \cdot M_{j,0})) \text{ and } (\rho_{\mathsf{prog}_j,0}, (\rho_{\mathsf{prog}_j,0} \cdot M_{j,0}))$$

We note that this encoding, relative to a random generator, is different from what was done in [24], and plays a crucial role in the security proof.

Randomizing The Matrices. As computed above, the encoded pairs clearly hide nothing: $M_{j,b}$ are binary matrices, and so they are completely revealed (via zero-testing). As a first step, we use the $\mathcal{NC}^1$ randomization technique (see [3, 31, 22]), as was done in [24] (however, unlike [24], we don't need to extend the matrix dimensions beyond 5×5). The idea is to generate a sequence of random matrices $\mathcal{Y}_j$ (over the ring R underlying the encoding scheme), and work with encodings of $\mathcal{N}_{j,b} = \mathcal{Y}_{j-1}^{-1} \cdot M_{j,b} \cdot \mathcal{Y}_j$ instead of the original $M_{j,b}$. This preserves the program's functional behavior, but each matrix, examined in isolation, becomes completely random. In fact, even if we take one matrix out of each pair, the joint distribution of these m matrices is uniformly random (see Section 2.1).

There is an obstacle here, because using the standard graded encoding interface, we can generate a random level 0 encoding of $\mathcal{Y}$, but we cannot derive $\mathcal{Y}^{-1}$ (in fact, this is not possible even for scalars). Indeed, to perform this step, [24] rely on the properties of a specific graded encoding instantiation. We propose a difference solution that works with any graded encoding scheme: instead of $\mathcal{Y}^{-1}$, we use the adjoint matrix $\mathcal{Z} = \mathsf{adj}(\mathcal{Y})$, which is composed of determinants of minors of $\mathcal{Y}$, and is therefore computable given the level 0 encoding of $\mathcal{Y}$. We know that $\mathcal{Y} \cdot \mathcal{Z} = \det(\mathcal{Y}) \cdot I$, which will be sufficient for our purposes.[7] Using the encoding scheme from [10, 11], in the j-th group prog_j we encode two pairs:

$$(\rho_{\mathsf{prog}_j,b}, (\rho_{\mathsf{prog}_j,b} \cdot \mathcal{N}_{j,b}))_{b \in \{0,1\}}, \text{ where } \mathcal{N}_{j,b} = \mathcal{Z}_{j-1} \cdot M_{j,b} \cdot \mathcal{Y}_j$$

To efficiently *evaluate* this program, we need an additional group, which we denote by chk. In this group we encode a random generator ρ_{chk}, and the element $(\rho_{\mathsf{chk}} \cdot (\prod_j \det(\mathcal{Y}_j)) \cdot \mathcal{Y}_m[1,1])$. We evaluate the branching program using the graded encoding scheme's zero-test feature, by checking whether:

$$\left((\rho_{\mathsf{prog}_1,x_{\ell(1)}} \mathcal{N}_{1,x_{\ell(1)}}) \cdots (\rho_{\mathsf{prog}_m,x_{\ell(m)}} \mathcal{N}_{m,x_{\ell(m)}}) \cdot \rho_{\mathsf{chk}}\right)[1,1] - \\ \rho_{\mathsf{prog}_1,x_{\ell(1)}} \cdots \rho_{\mathsf{prog}_m,x_{\ell(m)}} \cdot (\rho_{\mathsf{chk}} \cdot (\prod_j \det(\mathcal{Y}_j)) \cdot \mathcal{Y}_m[1,1]) = 0 \ .$$

This provides the required functionality, but it does not provide a secure construction.

setting of graded encoding schemes, the groups are replaced with indexed sets. See Section 2.4 for further details.

[7] We use here the fact that all matrices we work with are of constant dimension, and so we can compute the determinants of the minors in polynomial time while using only multilinear operations.

Enforcing Consistency. An obvious weakness of the above construction is that it does not verify *consistency.* For a variable x_i that appears multiple time in the program, the above scheme does not enforce that the same value will be used at all times. This will be handled, similarly to [24, 11], by adding *consistency check variables.* In each group grp that is "associated" with a variable x_i (so far, these only include groups of the form prog_j s.t. $\ell(j) = i$), the obfuscator generates two random variables $\beta_{\mathsf{grp},i,0}$ and $\beta_{\mathsf{grp},i,1}$, and multiplies the relevant variables. Namely, in group prog_j with $\ell(j) = i$, we provide encodings of

$$(\rho_{\mathsf{prog}_j,b}, (\rho_{\mathsf{prog}_j,b} \cdot \beta_{\mathsf{prog}_j,i,b} \cdot \mathcal{N}_{j,b}))_{b \in \{0,1\}}$$

To preserve functionality, we would like to choose the β variables so that the product of all zero-choices and the product of all one-choices are the same (one might even consider imposing a constraint that the product is 1). For clarity of exposition, we prefer the following solution: we use an additional auxiliary group cc_i for every variable x_i, such that

$$\beta_{\mathsf{cc}_i,\mathbf{0}} = \beta'_{\mathsf{cc}_i} \cdot \prod_{j:\ell(j)=i} \beta_{\mathsf{prog}_j,\mathbf{1}} \ ,$$

and vice versa (and β'_{cc_i} is the same for both cases). This guarantees that the product of all zero-choices, and the product of all one-choices, is the same. We denote this value by γ_i.

To preserve functionality, we multiply the element in the chk group by $\prod_i \gamma_i$. Now in the chk group we have encodings of:

$$(\rho_{\mathsf{chk}}, (\rho_{\mathsf{chk}} \cdot \prod_i \gamma_i \cdot (\prod_j \det(\mathcal{Y}_j)) \cdot \mathcal{Y}_m[1,1]))$$

Intuitively, it seems that this change renders inconsistent assignments useless: if, for some bit i of the input, the β values for i are not all taken according to the same value (0 or 1), then the constraint does not come into play. Therefore, the β values completely randomize these selected values.

One could postulate that the above construction is secure. In fact, we do not know of an explicit generic-model attack on this construction. Still, there are challenges to constructing a simulator. The crux of the difficulty is that an attacker might somehow efficiently produce a multilinear expression that corresponds to the evaluation of *multiple (super-polynomially many) consistent inputs* at the same time (or some function of super-polynomially many inputs: e.g. checking if the circuit accepts all of them simultaneously). This would break the obfuscator's security, since an (efficient) simulator cannot evaluate the function on super-polynomially many inputs.

Indistinguishability Obfuscation via Inefficient Simulation. If we allow a computationally unbounded simulator, then the above is not a problem. We show that the existence of a computationally unbounded black-box simulator implies indistinguishability obfuscation. In fact, the notions are equivalent both in the standard model and in the generic graded encoding scheme model. *Indistinguishability obfuscation* for $\mathcal{NC}^1$ therefore follows, and an indistinguishability obfuscator for $\mathcal{P}$ can be derived using the $\mathcal{NC}^1$ to $\mathcal{P}$ transformation of [24].

We note that the main conceptual difference that allows us to prove indistinguishability obfuscation for our construction, as opposed to [24]'s, is our use of the randomized ρ generators. This allows us, for any multilinear expression computed by the adversary, to isolate the relevant consistent inputs that affect the value of that expression.

Efficient Simulation and Virtual Black-Box Obfuscation. To get efficient black-box simulation (and virtual black-box security), we need to address the above difficulty. To do so, we build on the *randomizing sub-assignments* technique from [11]. Here, we use this technique to *bind the variables together* into triples. This done by adding $\binom{n}{3}$ additional groups, denoted bind_T, where $T \in \binom{[n]}{3}$ (i.e. one for each triple of variables). The group bind_T is associated with the triple of variables $\{i_1, i_2, i_3\} \in T$, and contains 8 pairs of encodings:

$$(\rho_{\mathsf{bind}_T,b_1b_2b_3}, (\rho_{\mathsf{bind}_T,b_1b_2b_3} \cdot \beta_{\mathsf{bind}_T,i_1,b_1} \cdot \beta_{\mathsf{bind}_T,i_2,b_2} \cdot \beta_{\mathsf{bind}_T,i_3,b_3}))_{b_1b_2b_3 \in \{0,1\}^3}$$

In evaluating the program on an input x, for each group bind_T, the evaluator chooses one of these 8 pairs according to the bits of $x_{|T}$, and uses it in computing the two products that go into the zero test (as above). The aforementioned consistency variables $\beta_{\mathsf{cc}_i,b}$ take the new β's into account, and are accordingly computed as

$$\beta_{\mathsf{cc}_i,0} = \beta'_{\mathsf{cc}_i} \cdot \prod_{j:\ell(j)=i} \beta_{\mathsf{prog}_j,1} \cdot \prod_{T:i\in T} \beta_{\mathsf{bind}_T,i,1} \ ,$$

and vice versa. The γ_i values are modified in the same way.

Intuitively, in order to evaluate the program, the adversary now needs not only to consistently choose the value of every single variable, but also to *jointly commit* to the values of each triple, consistently with its choices for the singleton variables. We show that if a polynomial adversary is able to produce an expression that corresponds to a sum of superpolynomially many consistent evaluations, then it can also evaluate a 3SAT formula on superpoynomially many values simultaneously, which contradicts the bounded speedup hypothesis (BSH, see Section 2.2).

A Taste of the Security Proof. The (high-level) intuition behind the security proof is as follows. In the idealized generic graded encoding scheme model, an adversary can only compute (via the encoding scheme) multilinear arithmetic circuits of the items encoded in the obfuscation. Moreover, the expansion of these multilinear circuits into a sum-of-monomials form, will only have one element from each group in each monomial (note that this expansion may be inefficient and the number of monomials can even be exponential). We call this a *cross-linear polynomial.*

The main challenge for simulation is "zero testing" of cross linear polynomials, given their circuit representation:[8] determining whether or not a polynomial f computed by the adversary takes value 0 on the items encoded in the obfuscation. We note that this is where we exploit the generic model—it allows us to reason about what functions the adversary is computing, and to assume that they have the restricted cross-linear form. We also note that zero-testing is a serious challenge, because the simulator does not know the joint distribution of the items encoded in the obfuscation (their joint distribution depends on the branching program C in its entirety). Due to space limitations, the full details on the simulator are deferred to the full version [12].

A cross-linear polynomial f computed by the adversary can be decomposed using monomials that only depend on the ρ variables in the construction outlined above. f must be a sum of such "ρ-monomials", each multiplied with a function of the other variables (the variables derived from the matrices of the branching program and the randomized elements used in the obfuscation). Because of the restricted structure of these ρ-monomials, they each implicitly specify an assignment to every program group prog_j (a bit value for the $\ell(j)$-th bit), every binding group bind_T (a triple of bit values for the input bits in T), and every consistency variable cc_i (a bit value for the i-th bit). We say that the assignment is *full and consistent*, if all of these groups are assigned appropriate values, and the value assigned to each bit i (0 or 1) is the same in throughout all the groups. We show that if a polynomial f computed by the adversary contains even a single such ρ-monomial that is *not full and consistent*, then it will not take 0 value (except with negligible probability over the obfuscator's coins), and thus it can be simulated. Further, if f contains only full and consistent ρ-monomials, the simulator can isolate each of these monomials, discover the associated full and consistent input assignment, and then zero-test f.

The main remaining concern, as hinted at above, is that f *might have super-polynomially many* full and consistent ρ-monomials. Isolating these monomials as described above would then take super-polynomial time (whereas we want a polynomial time black-box simulator). Intuitively, this corresponds to an adversary that can test some condition on the obfuscated circuit's behavior over super-polynomially many inputs (which the black-box simulator cannot do). Let $X \subseteq \{0,1\}^n$ denote the (super-polynomial) set of assignments associated with the above ρ-monomials. We show that given such f (even via black-box access), it is possible to test whether any given 3CNF formula Φ has a satisfying assignment in X. This yields *a worst-case* "super-polynomial speedup" in solving 3SAT. Since the alleged f is computable by a polynomial size arithmetic circuit, we get a contradiction to the Bounded Speedup Hypothesis.

This connection to solving 3SAT is proved by building on the "randomizing sub-assignment" technique from [11] and the groups bind_T. The intuition is as follows. The generic adversary implicitly specifies a polynomial-size arithmetic circuit that computes the function f. Recall that f has many full and consistent

[8] In fact, all we need is black-box access to the polynomial, and the *guarantee* that it is computable by a polynomial-size arithmetic circuit.

ρ-monomials, each associated with an input $x \in X \subseteq \{0,1\}^n$. For a given 3CNF formula Φ, we can compute a restriction of f by setting some of the variables $\rho_{\mathsf{bind}_T,i,\vec{v}}$ to 0, in a way that "zeroes out" every ρ-monomial associated with an input $x \in X$ that does not satisfy Φ, which is possible since the binding variables correspond exactly to all possible clauses in a 3CNF formula (see below). We then test to see whether or not the restricted polynomial (which has low degree) is identically 0, i.e. whether any of the ρ-monomials were *not* "zeroed out" by the above restriction. This tells us whether there exists $x \in X$ that satisfies Φ.

All that remains is to compute the restriction claimed above. For every clause in the 3CNF formula Φ, operating on variables $T = \{i_1, i_2, i_3\}$, there is an assignment $\vec{v} \in \{0,1\}^3$ that fails to satisfy the clause. For each such clause, we set the variable $\rho_{\mathsf{bind}_T,\vec{v}}$ to be 0. This effectively "zeroes out" all of the ρ-monomials whose associated assignments do not satisfy Φ (i.e., the ρ-monomials whose assignments simultaneously fail to satisfy *all* clauses in Φ).

The full construction appears in Section 3. The security proof appears in the full version [12], and is omitted from this extended abstract due to space limitations.

2 Preliminaries

For all $n, d \in \mathbb{N}$ we define $\binom{[n]}{d}$ to be the set of lexicographically ordered sets of cardinality d in $[n]$. More formally:

$$\binom{[n]}{d} = \left\{ \langle i_1, \ldots, i_d \rangle \in [n]^d : i_1 < \cdots < i_d \right\} .$$

Note that $\left|\binom{[n]}{d}\right| = \binom{n}{d}$.

For $\vec{x} \in \{0,1\}^n$ and $I = \langle i_1, \ldots, i_d \rangle \in \binom{[n]}{d}$, we let $\vec{x}_{|I} \in \{0,1\}^d$ denote the vector $\langle \vec{x}[i_1], \ldots, \vec{x}[i_d] \rangle$. We often slightly abuse notation when working with $\vec{s} = \vec{x}_{|I}$, and let $\vec{s}[i_j]$ denote the element $x[i_j]$ (rather than the i_jth element in $\vec{s}$).

2.1 Branching Programs and Randomizations

A width-5 length-m permutation branching program C for n-bit inputs is composed of: a sequence of pairs of permutations represented as 0/1 matrices $(M_{j,v} \in \{0,1\}^{5\times 5})_{j\in[m],v\in\{0,1\}}$, a labelling function $\ell : [m] \to [n]$, an accepting permutation Q_{acc}, and a rejecting permutation Q_{rej} s.t. $Q_{\mathsf{acc}}^T \cdot \vec{e}_1 = \vec{e}_1$ and $Q_{\mathsf{rej}}^T \cdot \vec{e}_1 = \vec{e}_k$ for $k \neq 1$. Without loss of generality, we assume that all permutation branching programs have the same accepting and rejecting permutations.

For an input $\vec{x} \in \{0,1\}$, taking $P = \prod_{j\in[m]} M_{j,\vec{x}[\ell(j)]}$, the program's output is 1 iff $P = Q_{\mathsf{acc}}$, and 0 iff $P = Q_{\mathsf{rej}}$. If P is not equal to either of these permutations, then the output is undefined (this will never be the case in any construction we use).

Barrington's Theorem [6] shows that any function in $\mathcal{NC}^1$, i.e. a function that can be computed by a circuit of depth d can be computed by a permutation branching program of length 4^d. Moreover, the theorem is constructive, and gives an algorithm that efficiently transforms any depth d circuit into a permutation branching program in time $2^{O(d)}$. This program is *oblivious*, in the sense that its labeling function is independent of the circuit C (and depends only on its depth). An immediate implication is that $\mathcal{NC}^1$ circuits, which have depth $d(n) = \log(n)$, can be transformed into polynomial length branching program, in polynomial time.

Theorem 2.1 (Barrington's Theorem [6]).
For any circuit depth d and input size n, there exists a length $m = 4^d$, a labeling function $\ell : [m] \to [n]$, an accepting permutation Q_{acc} and a rejecting permutation Q_{rej}, s.t. the following holds. For any circuit with input size n, depth d and fan-in 2, which computes a function f, there exists a permutation branching program of length m that uses the labeling function $\ell(\cdot)$, has accepting permutation Q_{acc} and rejecting permutation Q_{rej}, and computes the same function f.

The permutation branching program is computable in time poly(m) given the circuit description.

Randomized Branching Programs. Permutation branching programs are amenable to randomization techniques that have proved very useful in cryptography and complexity theory [3, 31, 22, 2]. The idea is to "randomize" each matrix pair while preserving the program's functional behavior. Specifically, taking p to be a large prime, for $i \in [m]$ multiply the i-th matrix pair (on its right) by a random invertible matrix $\mathcal{Y}_i \in \mathbb{Z}_p^{*5\times 5}$, and multiply the $(i+1)$-th pair by $\mathcal{Y}_i^{-1}$ (on its left). This gives a new branching program:

$$(\mathcal{N}_{j,v})_{j\in[m],v\in\{0,1\}} : \mathcal{N}_{j,v} = (\mathcal{Y}_{j-1}^{-1} \cdot M_{j,v} \cdot \mathcal{Y}_j)$$

(where we take $\mathcal{Y}_0^{-1}$ to be the identity). The new randomized program preserves functionality in the sense that intermediate matrices cancel out. For an input $\vec{x} \in \{0,1\}^n$, taking $P = \prod_j \mathcal{N}_{j,v}$, the program accepts $\vec{x}$ if $P = (Q_{\mathsf{acc}} \cdot \mathcal{Y}_m)$ (or, equivalently $P[1,1] = \mathcal{Y}_m[1,1]$) and rejects $\vec{x}$ if $P = (Q_{\mathsf{rej}} \cdot \mathcal{Y}_m)$ (or, equivalently, $P[1,1] = \mathcal{Y}_m[k,1]$, for $k \neq 1$). We note that there is a negligible probability of error due to the multiplication by $\mathcal{Y}_m$. In terms of randomization, one can see that for any assignment $y : [m] \to \{0,1\}$, the collection of matrices $(\mathcal{N}_{j,y(j)})_{j\in[m]}$ are uniformly random and independent invertible matrices. We note that this holds even if y is not a "consistent" assignment: for $j, j' \in [m] : \ell(j) = \ell(j') = i$, y can assign different values to j and j' (corresponding to an inconsistent assignment to the i-th bit of $\vec{x}$).

Implementing the randomization idea over graded encoding schemes (see Section 2.4) is not immediate, because we do not know an efficient procedure for computing inverses, and we also do not know how to sample random invertible matrices. To handle these difficulties, we utilize a variant of the above idea (see the discussion in Section 1.2).

Instead of $\mathcal{Y}_j^{-1}$, we use the adjoint matrix $\mathcal{Z}_j = \mathsf{adj}(\mathcal{Y}_j)$, which is composed of determinants of minors of $\mathcal{Y}$, and satisfies $\mathcal{Y} \cdot \mathcal{Z} = \det(\mathcal{Y}) \cdot I$. We take:

$$(\mathcal{N}_{j,v})_{j\in[m],v\in\{0,1\}} : \mathcal{N}_{j,v} = (\mathcal{Z}_{j-1} \cdot M_{j,v} \cdot \mathcal{Y}_j)$$

(where $\mathcal{Z}_0$ is again the identity matrix). Observe that for $\vec{x} \in \{0,1\}^n$, taking $P = \prod_j \mathcal{N}_{j,v}$, the program accepts $\vec{x}$ if $P[1,1] = ((\prod_{j\in[m-1]} \det(\mathcal{Y}_j)) \cdot \mathcal{Y}_m)[1,1]$ and rejects $\vec{x}$ if $P[1,1] = ((\prod_{j\in[m-1]} \det(\mathcal{Y}_j)) \cdot \mathcal{Y}_m)[k,1]$. It is not hard to see that this also preserves the randomization property from above. The only remaining subtlety is that we do not know how to pick a uniformly random invertible matrix (without being able to compute inverses). This is not a serious issue, because for a large enough prime p, we can simply sample uniformly random matrices in ${\mathbb{Z}_p}^{5\times 5}$, and their joint distribution will be statistically close to uniformly random *and invertible* matrices.

Lemma 2.2 (Randomized Branching Programs). *For security parameter $\lambda \in \mathbb{N}$, let p_λ be a prime in $[2^\lambda, 2^{\lambda+1}]$. Fix a length-$\ell$ permutation branching program, and let $y : [m] \to \{0,1\}$ be any assignment function. Let $(\mathcal{Y}_j)_{j\in[m]}$ be chosen uniformly at random from ${\mathbb{Z}_p}^{5\times 5}$, and for $j \in [m], v \in \{0,1\}$ take $\mathcal{N}_{j,v} = (\mathcal{Z}_{j-1} \cdot M_{j,v} \cdot \mathcal{Y}_j)$. (where $\mathcal{Z}_0$ is the identity matrix).*

Then the joint distribution of $(\mathcal{N}_{j,y(j)})_{j\in[m]}$ is negl(λ)-statistically close to uniformly random and independent.

2.2 The Bounded Speedup Hypothesis (BSH)

The *Bounded Speedup Hypothesis* was introduced in [11] as a strengthening of the exponential time hypothesis (ETH). Formally, the hypothesis is as follows.

Definition 2.3 ($\mathcal{X}$-3-SAT Solver). *Consider a family of sets $\mathcal{X} = \{X_n\}_{n\in\mathbb{N}}$ such that $X_n \subseteq \{0,1\}^n$. We say that an algorithm $\mathcal{A}$ is a $\mathcal{X}$-3-SAT solver if it solves the 3-SAT problem, restricted to inputs in $\mathcal{X}$. Namely, given a 3-CNF formula $\Phi : \{0,1\}^n \to \{0,1\}$, $\mathcal{A}$ finds whether there exists $x \in X_n$ such that $\Phi(x) = 1$.*

Assumption 2.4 (Bounded Speedup Hypothesis). *There exists a polynomial $p(\cdot)$, such that for any $\mathcal{X}$-3-SAT solver that runs in time $t(\cdot)$, the family of sets $\mathcal{X}$ is of size at most $p(t(\cdot))$.*

The plausibility of this assumption is discussed in [11, Appendix A], where it is shown that a quasi-polynomial variant of BSH follows from ETH. Further evidence comes from the field of parameterized complexity.

2.3 Obfuscation

Definition 2.5 (Virtual Black-Box Obfuscator [5]).

Let $\mathcal{C} = \{\mathcal{C}_n\}_{n \in \mathbb{N}}$ be a family of polynomial-size circuits, where $\mathcal{C}_n$ is a set of boolean circuits operating on inputs of length n. And let $\mathcal{O}$ be a PPTM algorithm, which takes as input an input length $n \in \mathbb{N}$, a circuit $C \in \mathcal{C}_n$, a security parameter $\lambda \in \mathbb{N}$, and outputs a boolean circuit $\mathcal{O}(C)$ (not necessarily in $\mathcal{C}$).

$\mathcal{O}$ is a (black-box) obfuscator *for the circuit family $\mathcal{C}$ if it satisfies:*

1. Preserving Functionality: *For every $n \in \mathbb{N}$, and every $C \in \mathcal{C}_n$, and every $\vec{x} \in \{0,1\}^n$, with all but negl(λ) probability over the coins of $\mathcal{O}$:*
$$(\mathcal{O}(C, 1^n, \lambda))(\vec{x}) = C(\vec{x})$$
2. Polynomial Slowdown: *For every $n, \lambda \in \mathbb{N}$ and $C \in \mathcal{C}$, the circuit $\mathcal{O}(C, 1^n, 1^\lambda)$ is of size at most poly$(|C|, n, \lambda)$.*
3. Virtual Black-Box: *For every (non-uniform) polynomial size adversary $\mathcal{A}$, there exists a (non-uniform) polynomial size simulator $\mathcal{S}$, such that for every $n \in \mathbb{N}$ and* for every *$C \in \mathcal{C}_n$:*
$$\Big| \Pr_{\mathcal{O},\mathcal{A}}[\mathcal{A}(\mathcal{O}(C, 1^n, 1^\lambda)) = 1] - \Pr_{\mathcal{S}}[\mathcal{S}^C(1^{|C|}, 1^n, 1^\lambda) = 1] \Big| = negl(\lambda)$$

Remark 2.6. A stronger notion of functionality, which also appears in the literature, requires that with overwhelming probability the obfuscated circuit is correct on *every input simultaneously.* We use the relaxed requirement that for every input (individually) the obfuscated circuit is correct with overwhelming probability (in both cases the probability is only over the obfuscator's coins). We note that our construction can be modified to achieve the stronger functionality property (by using a ring of sufficiently large size and the union bound).

Definition 2.7 (Indistinguishability Obfuscator [5]). *Let $\mathcal{C}$ be a circuit family and $\mathcal{O}$ a PPTM as in Definition 2.5. $\mathcal{O}$ is an* indistinguishability obfuscator *for $\mathcal{C}$ if it satisfies the preserving functionality and polynomial slowdown properties of Definition 2.5 with respect to $\mathcal{C}$, but the virtual black-box property is replaced with:*

3. Indistinguishable Obfuscation: *For every (non-uniform) polynomial size adversary $\mathcal{A}$, for every $n \in \mathbb{N}$ and* for every *$C_1, C_2 \in \mathcal{C}_n$ s.t. $|C_1| = |C_2|$ and $C_1 \equiv C_2$:*
$$\Big| \Pr_{\mathcal{O},\mathcal{A}}[\mathcal{A}(\mathcal{O}(C_1, 1^n, 1^\lambda)) = 1] - \Pr_{\mathcal{O},\mathcal{A}}[\mathcal{A}(\mathcal{O}(C_2, 1^n, 1^\lambda)) = 1] \Big| = negl(\lambda)$$

Definition 2.8 (iO2: Indistinguishability Obfuscator, Alternative Formulation). *Let $\mathcal{C}$ be a circuit family and $\mathcal{O}$ a PPTM as in Definition 2.5. $\mathcal{O}$ is an* indistinguishability obfuscator2 *(iO2) for $\mathcal{C}$ if it satisfies the preserving functionality and polynomial slowdown properties of Definition 2.5 with respect to $\mathcal{C}$, but the virtual black-box property is replaced with:*

3. Unbounded simulation: *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 *$C \in \mathcal{C}_n$:*
$$\Big| \Pr_{\mathcal{O},\mathcal{A}}[\mathcal{A}(\mathcal{O}(C, 1^n, 1^\lambda)) = 1] - \Pr_{\mathcal{S}}[\mathcal{S}^C(1^{|C|}, 1^n, 1^\lambda) = 1] \Big| = negl(\lambda)$$

Lemma 2.9. *Let $\mathcal{C}$ be a circuit family and $\mathcal{O}$ a PPTM as in Definition 2.5. Then $\mathcal{O}$ is* iO *if and only if it is* $\mathsf{iO2}$*.*

Proof. Assume that $\mathcal{O}$ is $\mathsf{iO2}$, let $\mathcal{A}$ be an adversary and let $C_1 \equiv C_2 \in \mathcal{C}_n$ s.t. $|C_1| = |C_2|$. Then by Definition 2.8 there exists a computationally unbounded $\mathcal{S}$ such that

$$\big| \Pr_{\mathcal{O},\mathcal{A}}[\mathcal{A}(\mathcal{O}(C_1, 1^n, 1^\lambda)) = 1] - \Pr_{\mathcal{S}}[\mathcal{S}^{C_1}(1^{|C_1|}, 1^n, 1^\lambda) = 1]\big| = negl(\lambda)$$

and also

$$\big| \Pr_{\mathcal{O},\mathcal{A}}[\mathcal{A}(\mathcal{O}(C_2, 1^n, 1^\lambda)) = 1] - \Pr_{\mathcal{S}}[\mathcal{S}^{C_2}(1^{|C_2|}, 1^n, 1^\lambda) = 1]\big| = negl(\lambda) \ .$$

However, since $C_1 \equiv C_2$, then $|C_1| = |C_2|$ and an oracle to C_1 is identical to an oracle to C_2, and in particular

$$\Pr_{\mathcal{S}}[\mathcal{S}^{C_1}(1^{|C_1|}, 1^n, 1^\lambda) = 1] = \Pr_{\mathcal{S}}[\mathcal{S}^{C_2}(1^{|C_2|}, 1^n, 1^\lambda) = 1] \ .$$

The triangle inequality immediately implies that

$$\big| \Pr_{\mathcal{O},\mathcal{A}}[\mathcal{A}(\mathcal{O}(C_1, 1^n, 1^\lambda)) = 1] - \Pr_{\mathcal{O},\mathcal{A}}[\mathcal{A}(\mathcal{O}(C_2, 1^n, 1^\lambda)) = 1]\big| = negl(\lambda) \ ,$$

and therefore $\mathcal{O}$ is iO.

In the opposite direction, let $\mathcal{O}$ be iO and let $\mathcal{A}$ be an adversary. We define the following simulator $\mathcal{S}^C(1^{|C|}, 1^n, 1^\lambda)$. The simulator first queries the oracle C on all $x \in \{0,1\}^n$, thus obtaining its truth table. Then it performs exhaustive search over all C' of size $1^{|C|}$ and finds $C' \in \mathcal{C}$ such that $C' \equiv C$ (this can be tested using the truth table). Finally the simulator outputs $\mathcal{A}(\mathcal{O}(C', 1^n, 1^\lambda))$.

By definition we have that

$$\big| \Pr_{\mathcal{O},\mathcal{A}}[\mathcal{A}(\mathcal{O}(C, 1^n, 1^\lambda)) = 1] - \Pr_{\mathcal{S}}[\mathcal{S}^{C}(1^{|C|}, 1^n, 1^\lambda) = 1]\big| = \\ \big| \Pr_{\mathcal{O},\mathcal{A}}[\mathcal{A}(\mathcal{O}(C, 1^n, 1^\lambda)) = 1] - \Pr_{\mathcal{O},\mathcal{A}}[\mathcal{A}(\mathcal{O}(C', 1^n, 1^\lambda)) = 1]\big| \ ,$$

and since $C \equiv C'$ and $\mathcal{O}$ is iO, this quantity is negligible and $\mathsf{iO2}$ follows.

2.4 Graded Encoding Schemes

We begin with the definition of a graded encoding scheme, due to Garg, Gentry and Halevi [23]. While their construction is very general, for our purposes a more restricted setting is sufficient as defined below.

Definition 2.10 (τ-Graded Encoding Scheme [23]). *A τ-encoding scheme for an integer $\tau \in \mathbb{N}$ and ring R, is a collection of sets $\mathcal{S} = \{S_{\vec{v}}^{(\alpha)} \subset \{0,1\}^* : \vec{v} \in \{0,1\}^\tau, \alpha \in R\}$ with the following properties:*

1. *For every index $\vec{v} \in \{0,1\}^\tau$, the sets $\{S_{\vec{v}}^{(\alpha)} : \alpha \in R\}$ are disjoint, and so they are a partition of the indexed set $S_{\vec{v}} = \bigcup_{\alpha \in R} S_{\vec{v}}^{(\alpha)}$.*
   In this work, for a 5×5 matrix $\mathcal{Y}$, we use $S_{\vec{v}}^{(\mathcal{Y})}$ to denote the set of 5×5 matrices where for all $i, j \in [5]$, the matrix's $[i,j]$-th entry contains an element in $S_{\vec{v}}^{(\mathcal{Y}[i,j])}$.
2. *There are binary operations "$+$" and "$-$" such that for all $\vec{v} \in \{0,1\}^\tau$, $\alpha_1, \alpha_2 \in R$ and for all $u_1 \in S_{\vec{v}}^{(\alpha_1)}$, $u_2 \in S_{\vec{v}}^{(\alpha_2)}$:*

$$u_1 + u_2 \in S_{\vec{v}}^{(\alpha_1+\alpha_2)} \quad \textit{and} \quad u_1 - u_2 \in S_{\vec{v}}^{(\alpha_1-\alpha_2)} \ ,$$

   where $\alpha_1 + \alpha_2$ and $\alpha_1 - \alpha_2$ are addition and subtraction in R.
3. *There is an associative binary operation "$\times$" such that for all $\vec{v}_1, \vec{v}_2 \in \{0,1\}^\tau$ such that $\vec{v}_1 + \vec{v}_2 \in \{0,1\}^\tau$, for all $\alpha_1, \alpha_2 \in R$ and for all $u_1 \in S_{\vec{v}_1}^{(\alpha_1)}$, $u_2 \in S_{\vec{v}_2}^{(\alpha_2)}$, it holds that*

$$u_1 \times u_2 \in S_{\vec{v}_1+\vec{v}_2}^{(\alpha_1 \cdot \alpha_2)},$$

   where $\alpha_1 \cdot \alpha_2$ is multiplication in R.

In this work, the ring R will always be $\mathbb{Z}_p$ for a prime p.

For the reader who is familiar with [23], we note that the above is the special case of their construction, in which we consider only binary index vectors (in the [23] notation, this corresponds to setting $\kappa = 1$), and we construct our encoding schemes to be *asymmetric* (as will become apparent below when we define our zero-test index $\mathbf{vzt} = \vec{1}$).

Definition 2.11 (Efficient Procedures for τ-Graded Encoding Scheme [23]). *We consider τ-graded encoding schemes (see above) where the following procedures are efficiently computable.*

- *Instance Generation:* $\mathsf{InstGen}(1^\lambda, 1^\tau)$ *outputs the set of parameters params, a description of a τ-Graded Encoding Scheme. (Recall that we only consider Graded Encoding Schemes over the set indices $\{0,1\}^\tau$, with zero testing in the set $S_{\vec{1}}$). In addition, the procedure outputs a subset evparams $\subset$ params that is sufficient for computing addition, multiplication and zero testing[9] (but possibly insufficient for encoding or for randomization).*
- *Ring Sampler:* $\mathsf{samp}(params)$ *outputs a "level zero encoding" $a \in S_0^{(\alpha)}$ for a nearly uniform $\alpha \in_R R$.*
- *Encode and Re-Randomize:*[10] $\mathsf{encRand}(params, i, a)$ *takes as input an index $i \in [\tau]$ and $a \in S_0^{(\alpha)}$, and outputs an encoding $u \in S_{\vec{e}_i}^{(\alpha)}$, where the distribution of u is (statistically close to being) only dependent on α and not otherwise dependent of a.*

[9] The "zero testing" parameter **pzt** defined in [23] is a part of *evparams*.

[10] This functionality is not explicitly provided by [23], however it can be obtained by combining their encoding and re-randomization procedures.

– *Addition and Negation:* $\mathsf{add}(evparams, u_1, u_2)$ *takes* $u_1 \in S_{\vec{v}}^{(\alpha_1)}, u_2 \in S_{\vec{v}}^{(\alpha_2)}$, *and outputs* $w \in S_{\vec{v}}^{(\alpha_1+\alpha_2)}$. *(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,* $\mathsf{negate}(evparams, u_1) \in S_{\vec{v}}^{(-\alpha_1)}$.
– *Multiplication:* $\mathsf{mult}(evparams, u_1, u_2)$ *takes* $u_1 \in S_{\vec{v}_1}^{(\alpha_1)}, u_2 \in S_{\vec{v}_2}^{(\alpha_2)}$. *If* $\vec{v}_1 + \vec{v}_2 \in \{0,1\}^\tau$ *(i.e. every coordinate in* $\vec{v}_1 + \vec{v}_2$ *is at most 1), then* mult *outputs* $w \in S_{\vec{v}_1+\vec{v}_2}^{(\alpha_1 \cdot \alpha_2)}$. *Otherwise,* mult *outputs* $\perp$. *We often use the notation* $u_1 \times u_2$ *to denote this operation when evparams is clear from the context.*
– *Zero Test:* $\mathsf{isZero}(evparams, u)$ *outputs 1 if* $u \in S_{\vec{1}}^{(0)}$, *and 0 otherwise.*

In the [23, 20] 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* $O(\tau)$ *operations, which are sufficient for our purposes. We therefore ignore noise management throughout this manuscript. An additional subtle issue is that with negligible probability the initial noise may be too big. However this can be avoided by adding rejection sampling to* samp *and therefore ignored throughout the manuscript as well.*

As was done in [10, 11], our definition deviates from that of [23]. We define two sets of parameters *params* and *evparams*. While the former will be used by the obfuscator in our construction (and therefore will not be revealed to an external adversary), the latter will be used when evaluating an obfuscated program (and thus will be known to an adversary). When instantiating our definition, the guideline is to make *evparams* minimal so as to give the least amount of information to the adversary. In particular, in the known candidates [23, 20], *evparams* only needs to contain the zero-test parameter **pzt** (as well as the global modulus).

2.5 The Generic Graded Encoding Scheme Model

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, which was previously used in [10], and is analogous to the *generic group model* (see Shoup [38] and Maurer [33]). In this model, an algorithm/adversary $\mathcal{A}$ can only interact with the graded encoding scheme via oracle calls to the add, mult, and isZero operations from Definition 2.11. Note that, in particular, we only allow access to the operations that can be run using *evparams*. To the best of our knowledge, non-generic attacks on known schemes, e.g. [23], require use of *params* and cannot be mounted when only *evparams* is given.

We use $\mathcal{G}$ to denote an oracle that answers adversary calls. The oracle operates as follows: for each index $\vec{v} \in \{0,1\}^\tau$, the elements of the indexed set $S_{\vec{v}} = \bigcup_{\alpha \in R} S_{\vec{v}}^{(\alpha)}$ are arbitrary binary strings. The adversary $\mathcal{A}$ can manipulate these strings using oracle calls (via $\mathcal{G}$) to the graded encoding scheme's functionalities. For example, the adversary can use $\mathcal{G}$ to perform an add call: taking strings $s_1 \in S_{\vec{v}}^{(\alpha_1)}, s_2 \in S_{\vec{v}}^{(\alpha_2)}$, encoding indexed ring elements $(\vec{v}, \alpha_1), (\vec{v}, \alpha_2)$ (respectively), and obtaining a string $s \in S_{\vec{v}}^{(\alpha_1+\alpha_2)}$, encoding the indexed ring element $(\vec{v}, (\alpha_1 + \alpha_2))$.

We say that $\mathcal{A}$ is a generic algorithm (or adversary) for a problem on graded encoding schemes (e.g. for computing a moral equivalent of discreet log), if it can accomplish this task with respect to *any* oracle representing a graded encoding scheme, see below.

In the add example above, there may be many strings/encodings in the set $S_{\vec{v}}^{(\alpha_1+\alpha_2)}$. One immediate question is *which* of these elements should be returned by the call to add. In our abstraction, for each $\vec{v} \in \{0,1\}^\tau$ and $\alpha \in R$, $\mathcal{G}$ always uses a *single unique encoding* of the indexed ring element $(\vec{v}, \alpha)$. I.e. the set $S_{\vec{v}}^{\alpha}$ is a singleton. Thus, the representation of items in the graded encoding scheme is given by a map $\sigma(\vec{v}, \alpha)$ from $\vec{v} \in \{0,1\}^\tau$ and $\alpha \in R$, to $\{0,1\}^*$. We restrict our attention to the case where this mapping has polynomial blowup.

Remark 2.12 (Unique versus Randomized Representation).
We note that the known candidates of secure graded encoding schemes [23, 20] *do not provide unique encodings*: their encodings are probabilistic. Nonetheless, in the generic graded encoding scheme abstraction we find it helpful to restrict our attention to schemes with unique encodings. For the purposes of proving security against generic adversaries, this makes sense: a generic adversary should work for *any* implementation of the oracle $\mathcal{G}$, and in particular also for an implementation that uses unique encodings.

Moreover, our perspective is that unique encodings are more "helpful" to an adversary than randomized encodings: a unique encoding gives the adversary the additional power to "automatically" check whether two encodings are of the same indexed ring element (without consulting the oracle). Thus, we prefer to prove security against generic adversaries even for unique representations.

It is important to note that the set of legal encodings may be very sparse within the set of images of σ, and indeed this is the main setting we will consider when we study the generic model. In this case, the only way for $\mathcal{A}$ to obtain a valid representation of any element in any graded set is via calls to the oracle (except with negligible probability). Finally, we note that if oracle calls contain invalid operators (e.g. the input is not an encoding of an element in any graded set, the inputs to add are not in the same graded set, etc.), then the oracle returns $\perp$.

Random Graded Encoding Scheme Oracle. We focus on a particular randomized oracle: the random generic encoding scheme (GES) oracle $\mathcal{RG}$. $\mathcal{RG}$ operates as follows: for each indexed ring element (with index $\vec{v} \in \{0,1\}^\tau$ and ring element $\sigma \in R$), its encoding is of length $\ell = (|\tau| \cdot \log |R| \cdot poly(\lambda))$ (where $|\tau|$ is the bit representation length of τ). The encoding of each indexed ring element is a uniformly random string of length ℓ. In particular, this implies that the only way that $\mathcal{A}$ can obtain valid encodings is by calls to the oracle $\mathcal{RG}$ (except with negligible probability).

The definitions of secure obfuscation in the random GES model are as follows.

Definition 2.13 (Virtual Black-Box in the Random GES Model). *Let $\mathcal{C} = \{\mathcal{C}_n\}_{n\in\mathbb{N}}$ be a family of circuits and $\mathcal{O}$ a PPTM as in Definition 2.5. A generic algorithm $\mathcal{O}^{\mathcal{RG}}$ is an obfuscator* in the random generic encoding scheme model*, if it satisfies the functionality and polynomial slowdown properties of Definition 2.5 with respect to $\mathcal{C}$ and to* any *GES oracle $\mathcal{RG}$, but the virtual black-box property is replaced with:*

3. Virtual Black-Box in the Random GES Model: *For every (non-uniform) polynomial size* generic *adversary $\mathcal{A}$, there exists a (non-uniform) generic polynomial size simulator $\mathcal{S}$, such that for every $n \in \mathbb{N}$ and every $C \in \mathcal{C}_n$:*

$$\left|\left(\Pr_{\mathcal{RG},\mathcal{O},\mathcal{A}}[\mathcal{A}^{\mathcal{RG}}(\mathcal{O}^{\mathcal{RG}}(C,1^n,1^\lambda))] = 1\right) - \left(\Pr_{\mathcal{S}}[\mathcal{S}^C(1^{|C|},1^n,1^\lambda)] = 1\right)\right| = negl(\lambda)$$

Remark 2.14. We remark that it makes sense to allow $\mathcal{S}$ to access the oracle $\mathcal{RG}$. However, this is in not really necessary. The reason is that $\mathcal{RG}$ can be implemented in polynomial time (as described below), and therefore $\mathcal{S}$ itself can implement $\mathcal{RG}$.

Definition 2.15 (GiO: Indistinguishability Obfuscator in the Random GES Model). *Let $\mathcal{C} = \{\mathcal{C}_n\}_{n\in\mathbb{N}}$ be a family of circuits and $\mathcal{O}$ a PPTM as in Definition 2.5. A generic algorithm $\mathcal{O}^{\mathcal{RG}}$ is an indistinguishability obfuscator* in the random generic encoding scheme model*, if it satisfies the functionality and polynomial slowdown properties of Definition 2.5 with respect to $\mathcal{C}$ and to* any *GES oracle $\mathcal{RG}$, but the virtual black-box property is replaced with:*

3. Indistinguishable Obfuscation in the Random GES Model: *For every (non-uniform) polynomial size* generic *adversary $\mathcal{A}$, for every $n \in \mathbb{N}$ and* for every *$C_1, C_2 \in \mathcal{C}_n$ s.t. $C_1 \equiv C_2$ and $|C_1| = |C_2|$:*

$$\left|\left(\Pr_{\mathcal{RG},\mathcal{O},\mathcal{A}}[\mathcal{A}^{\mathcal{RG}}(\mathcal{O}^{\mathcal{RG}}(C_1,1^n,1^\lambda))] = 1\right) - \Pr_{\mathcal{RG},\mathcal{O},\mathcal{A}}[\mathcal{A}^{\mathcal{RG}}(\mathcal{O}^{\mathcal{RG}}(C_2,1^n,1^\lambda))] = 1\right| = negl(\lambda)$$

Definition 2.16 (GiO2: Indistinguishability Obfuscator in the Random GES Model, Alternative Formulation). *Let $\mathcal{C} = \{\mathcal{C}_n\}_{n\in\mathbb{N}}$ be a family of circuits and $\mathcal{O}$ a PPTM as in Definition 2.5. A generic algorithm $\mathcal{O}^{\mathcal{RG}}$ is an* indistinguishability obfuscator2 in the random generic encoding scheme model *(GiO2), if it satisfies the functionality and polynomial slowdown properties of Definition 2.5 with respect to $\mathcal{C}$ and to* any *GES oracle $\mathcal{RG}$, but the virtual black-box property is replaced with:*

3. Indistinguishable Obfuscation2 in the Random GES Model: *For every (non-uniform) polynomial size* generic *adversary $\mathcal{A}$, there exists a (possibly computationally unbounded) simulator $\mathcal{S}$, such that for every $n \in \mathbb{N}$ and every $C \in \mathcal{C}_n$:*

$$\left|\left(\Pr_{\mathcal{RG},\mathcal{O},\mathcal{A}}[\mathcal{A}^{\mathcal{RG}}(\mathcal{O}^{\mathcal{RG}}(C,1^n,1^\lambda))] = 1\right) - \left(\Pr_{\mathcal{S}}[\mathcal{S}^C(1^{|C|},1^n,1^\lambda)] = 1\right)\right| = negl(\lambda)$$

The following lemma asserts the equivalence between GiO and $\mathsf{GiO2}$ in the generic model. The proof is identical to that of Lemma 2.9 (note that $\mathcal{S}$ does not need access to $\mathcal{RG}$).

Lemma 2.17. *Let $\mathcal{C}$ be a circuit family and $\mathcal{O} = \mathcal{O}^{\mathcal{RG}}$ an oracle PPTM as in Definition 2.5. Then $\mathcal{O}$ is GiO if and only if it is $\mathsf{GiO2}$.*

Online Random GES Oracle. In our proof, we will use the property that the oracle $\mathcal{RG}$ can be approximated to within negligible statistical distance 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 $(\vec{v}, \alpha, \mathsf{label}_{\vec{v},\alpha})$, where $\mathsf{label}_{\vec{v},\alpha} \in \{0,1\}^{\ell}$ is the representation of $S_{\vec{v}}^{(\alpha)}$ in $\mathcal{RG}$ (the table is initially empty). Every time $\mathcal{RG}$ 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 $(\vec{v}, \alpha)$ to perform the operation (if the operands are not in the table, $\mathcal{RG}$ returns $\perp$). Whenever $\mathcal{RG}$ needs to return a value $S_{\vec{v}}^{(\alpha)}$, it checks whether $(\vec{v}, \alpha)$ is already in the table, and if so returns the appropriate $\mathsf{label}_{\vec{v},\alpha}$. 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 $\mathcal{RG}$ 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.

3 Obfuscating $\mathcal{NC}^1$

See Section 1.2 for an overview of the construction. We proceed with a formal description of the obfuscator. Functionality follows by construction, virtual black-box security is only stated, and the proof is deferred to the full version [12].

Obfuscator $\mathsf{NC^1Obf}$, on input $(1^{\lambda}, 1^{n}, C = (\ldots, (M_{j,0}, M_{j,1}), \ldots)_{j\in[m]})$

Input: Security parameter λ; Number of input variables n; Oblivious permutation branching program C (with labeling function ℓ), where $(M_{j,b})_{j\in[m], b\in\{0,1\}}$ are 5×5 permutation matrices. Let $Q_{\mathsf{acc}}, Q_{\mathsf{rej}}$ be the accepting and rejecting permutations for C (see Section 2.1).

Output: Obfuscated program for C.

Execution:

1. **Generate asymmetric encoding scheme.**
   Generate $(params, evparams) \leftarrow \mathsf{InstGen}(1^{\lambda}, 1^{\tau})$, where $\tau = m+n+\binom{n}{3}+1$. Namely, we have τ level-1 groups. As explained above, we denote these groups as follows:

– Groups $1, \dots, m$ are related to the execution of the branching program and are denoted $\mathsf{prog}_1, \dots, \mathsf{prog}_m$.

– Groups $m+1, \dots, m+n$ are related to consistency check and are denoted $\mathsf{cc}_1, \dots, \mathsf{cc}_n$.

– Groups $m+n+1, \dots, m+n+\binom{n}{3}$ are used to bind triples of variables and are denoted bind_T, for $T \in \binom{[n]}{3}$.

– Lastly, the group $m+n+\binom{n}{3}+1$ is the check group and is denoted chk.

We let $L(i)$ denote the set of groups that are related to the ith variable:

$$L(i) = \{\mathsf{prog}_j : i = \ell(j)\} \cup \{\mathsf{bind}_T : i \in T\} \cup \{\mathsf{cc}_i\} \ .$$

We let L denote the set of all groups: $L = \cup_{i \in [n]} L(i) \cup \mathsf{chk}$.

2. **Generate consistency check variables.**
For all $i \in [n]$:
(a) for each $\mathsf{grp} \in (L(i) \setminus \{\mathsf{cc}_i\})$ and $v \in \{0,1\}$: $b_{\mathsf{grp},i,v} \leftarrow \mathsf{samp}(params) \in S_0^{(\beta_{\mathsf{grp},i,v})}$. [11]

(b) $b'_{\mathsf{cc}_i} \leftarrow \mathsf{samp}(params) \in S_0^{(\beta'_{\mathsf{cc}_i})}$
for $v \in \{0,1\}$: $b_{\mathsf{cc}_i,i,v} \leftarrow b'_{\mathsf{cc}_i} \times \left(\prod_{\mathsf{grp} \in (L(i) \setminus \{\mathsf{cc}_i\})} b_{\mathsf{grp},i,(1-v)}\right) \in S_0^{(\beta_{\mathsf{cc}_i,v})}$

(c) $c_i \leftarrow \prod_{\mathsf{grp} \in L(i)} b_{\mathsf{grp},i,0} \in S_0^{(\gamma_i)}$, where

$$\gamma_i = \prod_{\mathsf{grp} \in L(i)} \beta_{\mathsf{grp},i,0} = \prod_{\mathsf{grp} \in L(i)} \beta_{\mathsf{grp},i,1} \ .$$

3. **Generate randomizing matrices.**
For each $j \in [m]$:
Sample $Y_j \leftarrow \mathsf{samp}(params)^{5\times 5} \in S_0^{(\mathcal{Y}_j)}$,
and compute $Z_j = \mathsf{adj}(Y_j) \in S_0^{(\mathcal{Z}_j)}$, s.t. $\mathcal{Y}_j \cdot \mathcal{Z}_j = \det(\mathcal{Y}_j) \cdot I$
Sample $Z_0 \leftarrow \mathsf{samp}(params)^{5\times 5} \in S_0^{(\mathcal{Z}_0)}$

4. **Encode elements in program groups (prog).**
For each $j \in [m]$ and $v \in \{0,1\}$:
$D_{\mathsf{prog}_j,v} \leftarrow b_{\mathsf{prog}_j,v} \cdot (Z_{j-1} \times M_{j,v} \times Y_j) \in S_0^{(\mathcal{D}_{j,v})}$, where $\mathcal{D}_{j,v} = (\beta_{\mathsf{prog}_j,v} \cdot \underbrace{(\mathcal{Z}_{j-1} \cdot M_{j,v} \cdot \mathcal{Y}_j)}_{\mathcal{N}_{j,v}})$
$r_{\mathsf{prog}_j,v} \leftarrow \mathsf{samp}(params) \in S_0^{(\rho_{\mathsf{prog}_j,v})}$,
$K_{\mathsf{prog}_j,v} \leftarrow (r_{\mathsf{prog}_j,v} \cdot D_{j,v}) \in S_0^{(\rho_{\mathsf{prog}_j,v} \cdot \mathcal{D}_{j,v})}$

[11] For notational convenience, we drop i when it is uniquely defined by grp. E.g., we use $\beta_{\mathsf{prog}_j,v}$ to refer to $\beta_{\mathsf{prog}_j,\ell(j),v}$.

$w_{\mathsf{prog}_j,v} \leftarrow \mathsf{encRand}(params, \mathsf{prog}_j, r_{\mathsf{prog}_j,v}) \in S^{(\rho_{\mathsf{prog}_j,v})}_{\vec{e}_{\mathsf{prog}_j}}$,
$U_{\mathsf{prog}_j,v} \leftarrow \mathsf{encRand}(params, \mathsf{prog}_j, K_{\mathsf{prog}_j,v}) \in S^{(\rho_{\mathsf{prog}_j,v}\cdot\mathcal{D}_{j,v})}_{\vec{e}_{\mathsf{prog}_j}}$

5. **Encode elements in consistency check groups (cc).**
For each $i \in [n]$ and $v \in \{0,1\}$:
$d_{\mathsf{cc}_i,v} \leftarrow b_{\mathsf{cc}_i,\vec{v}[i]} \in S^{(\delta_{\mathsf{cc}_i,v})}_0$, where $\delta_{\mathsf{cc}_i,v} = \beta_{\mathsf{cc}_i,v}$.
$r_{\mathsf{cc}_i,v} \leftarrow \mathsf{samp}(params) \in S^{(\rho_{\mathsf{cc}_i,v})}_0$, $q_{\mathsf{cc}_i,v} \leftarrow (r_{\mathsf{cc}_i,v}\cdot d_{\mathsf{cc}_i,v}) \in S^{(\rho_{\mathsf{cc}_i,v}\cdot\delta_{\mathsf{cc}_i,v})}_0$
$w_{\mathsf{cc}_i,v} \leftarrow \mathsf{encRand}(params, \mathsf{cc}_i, r_{\mathsf{cc}_i,v}) \in S^{(\rho_{\mathsf{cc}_i,v})}_{\vec{e}_{\mathsf{cc}_i}}$
$u_{\mathsf{cc}_i,v} \leftarrow \mathsf{encRand}(params, \mathsf{cc}_i, q_{\mathsf{cc}_i,v}) \in S^{(\rho_{\mathsf{cc}_i,v}\cdot\delta_{\mathsf{cc}_i,v})}_{\vec{e}_{\mathsf{cc}_i}}$

6. **Encode elements in binding groups (bind).**
For each $T \in \binom{[n]}{3}$ and $\vec{v} \in \{0,1\}^3$:
$d_{\mathsf{bind}_T,\vec{v}} \leftarrow (\prod_{i\in T} b_{\mathsf{bind}_T,i,\vec{v}[i]}) \in S^{(\delta_{\mathsf{bind}_T,\vec{v}})}_0$, where $\delta_{\mathsf{bind}_T,\vec{v}} = \prod_{i\in T}\beta_{\mathsf{bind}_T,i,\vec{v}[i]}$
$r_{\mathsf{bind}_T,\vec{v}} \leftarrow \mathsf{samp}(params) \in S^{(\rho_{\mathsf{bind}_T,\vec{v}})}_0$,
$q_{\mathsf{bind}_T,\vec{v}} \leftarrow (r_{\mathsf{bind}_T,\vec{v}}\cdot d_{\mathsf{bind}_T,\vec{v}}) \in S^{(\rho_{\mathsf{bind}_T,\vec{v}}\cdot\delta_{\mathsf{bind}_T,\vec{v}})}_0$
$w_{\mathsf{bind}_T,\vec{v}} \leftarrow \mathsf{encRand}(params, \mathsf{bind}_T, r_{\mathsf{bind}_T,\vec{v}}) \in S^{(\rho_{\mathsf{bind}_T,\vec{v}})}_{\vec{e}_{\mathsf{bind}_T}}$,
$u_{\mathsf{bind}_T,\vec{v}} \leftarrow \mathsf{encRand}(params, \mathsf{bind}_T, q_{\mathsf{bind}_T,\vec{v}}) \in S^{(\rho_{\mathsf{bind}_T,\vec{v}}\cdot\delta_{\mathsf{bind}_T,\vec{v}})}_{\vec{e}_{\mathsf{bind}_T}}$

7. **Encode elements in last group (chk).**
$d_{\mathsf{chk}} \leftarrow \left((\prod_{i\in[n]} c_i)\cdot(\prod_{j\in[m-1]}\det(Y_j))\cdot Z_0\cdot Y_m\right)[1,1] \in S^{(\delta_{\mathsf{chk}})}_0$,
where $\delta_{\mathsf{chk}} = \left((\prod_{i\in[n]}\gamma_i)\cdot(\prod_{j\in[m-1]}\det(\mathcal{Y}_j))\cdot\mathcal{Z}_0\cdot\mathcal{Y}_m\right)[1,1]$
$r_{\mathsf{chk}} \leftarrow \mathsf{samp}(params) \in S^{(\rho_{\mathsf{chk}})}_0$, $q_{\mathsf{chk}} \leftarrow r_{\mathsf{chk}}\cdot d_{\mathsf{chk}} \in S^{(\rho_{\mathsf{chk}}\cdot\delta_{\mathsf{chk}})}_0$
$w_{\mathsf{chk}} \leftarrow \mathsf{encRand}(params, \mathsf{chk}, r_{\mathsf{chk}}) \in S^{(\rho_{\mathsf{chk}})}_{\vec{e}_{\mathsf{chk}}}$,
$u_{\mathsf{chk}} \leftarrow \mathsf{encRand}(params, \mathsf{chk}, q_{\mathsf{chk}}) \in S^{(\rho_{\mathsf{chk}}\cdot\delta_{\mathsf{chk}})}_{\vec{e}_{\mathsf{chk}}}$

8. **Output.**
Output *evparams* and the obfuscation:

$$\Big(\{(w_{\mathsf{prog}_j,v}, U_{\mathsf{prog}_j,v})\}_{j\in[m],v\in\{0,1\}}, \{(w_{\mathsf{cc}_i,v}, u_{\mathsf{cc}_i,v})\}_{i\in[n],v\in\{0,1\}},$$
$$\{(w_{\mathsf{bind}_T,\vec{v}}, u_{\mathsf{bind}_T,\vec{v}})\}_{T\in(\binom{[n]}{3}),\vec{v}\in\{0,1\}^3}, (w_{\mathsf{chk}}, u_{\mathsf{chk}})\Big)$$

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

1. $\mathbf{t} \leftarrow \left(w_{\mathsf{chk}}\cdot(\prod_{j\in[m]} U_{\mathsf{prog}_j,x[\ell(j)]})\cdot(\prod_{T\in\binom{[n]}{3}} u_{\mathsf{bind}_T,x_{|T}})\cdot(\prod_{i\in[n]} u_{\mathsf{cc}_i,x[i]})\right)[1,1]$
2. $\mathbf{t}' \leftarrow \left(u_{\mathsf{chk}}\cdot(\prod_{j\in[m]} w_{\mathsf{prog}_j,x[\ell(j)]})\cdot(\prod_{T\in\binom{[n]}{3}} w_{\mathsf{bind}_T,x_{|T}})\cdot(\prod_{i\in[n]} w_{\mathsf{cc}_i,x[i]})\right)$
3. output the bit: $\mathsf{isZero}(evparams, (\mathbf{t}-\mathbf{t}'))$.

Virtual Black-Box Security. The following theorem states the security properties of our scheme. The proof is deferred to the full version [12].

Theorem 3.1. *Under the bounded speedup hypothesis,* $\mathsf{NC^1Obf}$ *is a virtual black box obfuscator in the random GES model for the class* $\mathcal{NC}^1$.

References

1. B. Adida and D. Wikström. How to shuffle in public. In S. P. Vadhan, editor, *TCC*, volume 4392 of *Lecture Notes in Computer Science*, pages 555–574. Springer, 2007.
2. B. Applebaum, Y. Ishai, and E. Kushilevitz. Cryptography in nc^0. *SIAM J. Comput.*, 36(4):845–888, 2006.
3. L. Babai. Trading group theory for randomness. In *STOC*, pages 421–429, 1985.
4. B. Barak, S. Garg, Y. T. Kalai, O. Paneth, and A. Sahai. Protecting obfuscation against algebraic attacks. Cryptology ePrint Archive, Report 2013/631, 2013. http://eprint.iacr.org/.
5. B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. P. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2):6, 2012. Preliminary version in CRYPTO 2001.
6. D. A. M. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. In J. Hartmanis, editor, *STOC*, pages 1–5. ACM, 1986. Full version in [7].
7. D. A. M. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *J. Comput. Syst. Sci.*, 38(1):150–164, 1989.
8. N. Bitansky and R. Canetti. On strong simulation and composable point obfuscation. In *CRYPTO*, pages 520–537, 2010.
9. D. Boneh and A. Silverberg. Applications of multilinear forms to cryptography. *IACR Cryptology ePrint Archive*, 2002:80, 2002.
10. Z. Brakerski and G. N. Rothblum. obfuscating conjunctions. In R. Canetti and J. A. Garay, editors, *CRYPTO (2)*, volume 8043 of *Lecture Notes in Computer Science*, pages 416–434. Springer, 2013. Full version in http://eprint.iacr.org/2013/471.
11. Z. Brakerski and G. N. Rothblum. Black-box obfuscation for d-CNFs. Cryptology ePrint Archive, 2013. Extended abstract in ITCS 2014.
12. Z. Brakerski and G. N. Rothblum. Virtual black-box obfuscation for all circuits via generic graded encoding. Cryptology ePrint Archive, Report 2013/563, 2013. Full version of this paper.
13. R. Canetti. Towards realizing random oracles: Hash functions that hide all partial information. In *CRYPTO*, pages 455–469, 1997.
14. R. Canetti and R. R. Dakdouk. Obfuscating point functions with multibit output. In *EUROCRYPT*, pages 489–508, 2008.
15. R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited (preliminary version). In J. S. Vitter, editor, *STOC*, pages 209–218. ACM, 1998. Full version in [16].
16. R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004.
17. R. Canetti, D. Micciancio, and O. Reingold. Perfectly one-way probabilistic hash functions (preliminary version). In J. S. Vitter, editor, *STOC*, pages 131–140. ACM, 1998.

18. R. Canetti, G. N. Rothblum, and M. Varia. Obfuscation of hyperplane membership. In *TCC*, pages 72–89, 2010.
19. R. Canetti and V. Vaikuntanathan. Obfuscating branching programs using black-box pseudo-free groups. Cryptology ePrint Archive, Report 2013/500, 2013. http://eprint.iacr.org/.
20. J.-S. Coron, T. Lepoint, and M. Tibouchi. Practical multilinear maps over the integers. In R. Canetti and J. A. Garay, editors, *CRYPTO (1)*, volume 8042 of *Lecture Notes in Computer Science*, pages 476–493. Springer, 2013.
21. Y. Dodis and A. Smith. Correcting errors without leaking partial information. In H. N. Gabow and R. Fagin, editors, *STOC*, pages 654–663. ACM, 2005.
22. U. Feige, J. Kilian, and M. Naor. A minimal model for secure computation (extended abstract). In *STOC*, pages 554–563, 1994.
23. S. Garg, C. Gentry, and S. Halevi. Candidate multilinear maps from ideal lattices. In T. Johansson and P. Q. Nguyen, editors, *EUROCRYPT*, volume 7881 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2013.
24. S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. Cryptology ePrint Archive, Report 2013/451, 2013. Extended abstract in FOCS 2013.
25. C. Gentry. Fully homomorphic encryption using ideal lattices. In M. Mitzenmacher, editor, *STOC*, pages 169–178. ACM, 2009.
26. S. Goldwasser and Y. T. Kalai. On the impossibility of obfuscation with auxiliary input. In *FOCS*, pages 553–562, 2005.
27. S. Goldwasser and G. N. Rothblum. On best-possible obfuscation. In S. P. Vadhan, editor, *TCC*, volume 4392 of *Lecture Notes in Computer Science*, pages 194–213. Springer, 2007.
28. D. Hofheinz, J. Malone-Lee, and M. Stam. Obfuscation for cryptographic purposes. *J. Cryptology*, 23(1):121–168, 2010.
29. S. Hohenberger, G. N. Rothblum, A. Shelat, and V. Vaikuntanathan. Securely obfuscating re-encryption. *J. Cryptology*, 24(4):694–719, 2011.
30. R. Impagliazzo and R. Paturi. Complexity of k-sat. In *IEEE Conference on Computational Complexity*, pages 237–240. IEEE Computer Society, 1999.
31. J. Kilian. Founding cryptography on oblivious transfer. In J. Simon, editor, *STOC*, pages 20–31. ACM, 1988.
32. B. Lynn, M. Prabhakaran, and A. Sahai. Positive results and techniques for obfuscation. In *EUROCRYPT*, pages 20–39, 2004.
33. U. . Maurer. Abstract models of computation in cryptography. In N. P. Smart, editor, *IMA Int. Conf.*, volume 3796 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2005.
34. M. Naor. On cryptographic assumptions and challenges. In D. Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 96–109. Springer, 2003.
35. R. Rivest, L. Adleman, and M. Dertouzos. On data banks and privacy homomorphisms. In *Foundations of Secure Computation*, pages 169–177. Academic Press, 1978.
36. R. Rothblum. On the circular security of bit-encryption. In *TCC*, pages 579–598, 2013.
37. A. Sahai and B. Waters. How to use indistinguishability obfuscation: Deniable encryption, and more. Cryptology ePrint Archive, Report 2013/454, 2013. http://eprint.iacr.org/.

38. V. Shoup. Lower bounds for discrete logarithms and related problems. In W. Fumy, editor, *EUROCRYPT*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266. Springer, 1997.
39. H. Wee. On obfuscating point functions. In H. N. Gabow and R. Fagin, editors, *STOC*, pages 523–532. ACM, 2005.