J Cryptol (2017) 30:289–320
DOI: 10.1007/s00145-015-9221-5

Journal of CRYPTOLOGY

Obfuscating Conjunctions

Zvika Brakerski*
Weizmann Institute of Science, Rehovot, Israel
zvika.brakerski@weizmann.ac.il

Guy N. Rothblum
Samsung Research America, Mountain View, CA, USA
rothblum@alum.mit.edu

Communicated by Ran Canetti

Received 15 August 2013
Online publication 23 November 2015

Abstract. We show how to securely obfuscate the class of *conjunction functions* (functions like $f(x_1, \ldots, x_n) = x_1 \wedge \neg x_4 \wedge \neg x_6 \wedge \cdots \wedge x_{n-2}$). Given any function in the class, we produce an obfuscated program which preserves the input–output functionality of the given function, but reveals nothing else. Our construction is based on multilinear maps, and can be instantiated using the recent candidates proposed by Garg, Gentry and Halevi (EUROCRYPT 2013) and by Coron et al. (CRYPTO 2013). We show that the construction is secure when the conjunction is drawn from a distribution, under mild conditions on the distribution. Security follows from multilinear entropic variants of the Diffie–Hellman assumption. We conjecture that our construction is secure for *any* conjunction, regardless of the distribution from which it is drawn. We offer supporting evidence for this conjecture, proving that our obfuscator is secure for any conjunction against *generic adversaries*.

Keywords. Program obfuscation, Graded encoding schemes, Conjunctions.

1. Introduction

Code obfuscation is the problem of compiling a computer program so as to make it unintelligible to an adversary, or impossible to reverse-engineer, while preserving its input–output functionality. Obfuscation has been of long-standing interest to both the cryptography and security communities. However, despite the importance of the problem, and its many exciting applications, very few techniques or effective heuristics are known. In particular, the theoretical study of the problem (in the "virtual black box model" [6]) led to a handful of known constructions, which apply to very limited classes

* Zvika Brakerski: Supported by a Simons Postdoctoral Fellowship and by DARPA.

of functions. These include the class of point functions, and extensions such as multi-point functions, "lockers" and constant-dimension hyperplanes.

In this work, we present an obfuscator for a new and different class: *conjunction functions*. These are functions that take n-bit strings as input and only accept if a subset of these bits are set to predefined values. Our construction relies on *(asymmetric) multilinear maps*, and is instantiated using the new candidate construction due to Garg et al. [21].

Previous Results The goal of an obfuscator is generating a program that preserves the functionality of the original program, but reveals nothing else. One commonly used formalization of this objective is "virtual black box" obfuscation, due to Barak et al. [6]. Our work uses this formalization , as well as alternative formalizations from subsequent works (see below).

In their work, Barak et al. [6] also proved the impossibility of general-purpose obfuscators (i.e., ones that work for any functionality) in the virtual black box model. This impossibility result was extended in [23]. While these negative results show serious limitations on the possibility of general-purpose obfuscation, they focus on specific functionalities, e.g., cryptographic functionalities with pseudo-entropy in the results of [23]. They do not rule out the possibility of obfuscating many natural functionalities.

Positive results on obfuscation focus 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 [10], several works have shown obfuscators for this family under various assumptions [16,29,35], as well as extensions [4,11]. Canetti et al. [17] 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 [19], vote mixing [2], and re-encryption [27]. Several works [10,16,26,27] relaxed the security requirement so that obfuscation only holds for a random choice of a program from the family, we will also use this relaxation for one of our results. A different relaxation, known as "best-possible obfuscation," which allows the obfuscation to leak non-black box information was presented in [24].

This Work: Obfuscating Conjunctions Our main contribution is a new obfuscator for *conjunctions*. A conjunction $C = (W, V)$ is a function on n bit inputs, specified by a set $W \subseteq [n]$ of "wildcard" entries, and a vector $V \in \{0,1\}^n$ of target values for non-wildcard entries. The conjunction accepts an input $\vec{x} \in \{0,1\}^n$ if for all $i \in ([n] \setminus W)$, $\vec{x}[i] = V[i]$, i.e., if for all non-wildcard entries in $\vec{x}$, their values equal those specified in V. We use the convention that if $W[i] = 1$ then $V[i] = 0$ (wildcard entries are ignored, so this does not affect the conjunction's functionality).

The class of conjunctions, while obviously quite limited, has a rich combinatorial and computational expressive power. They are studied in a multitude of settings throughout computer science (e.g., in learning theory [28]). One significant distinction from previous function classes for which obfuscators were known is that a conjunction may ignore some of its input bits (the wildcard entries). An obfuscator for conjunctions needs to produce a program that hides which bits are ignored and which ones are influential. Conjunctions are similar to prior work in that the set of accepting inputs can be sparse,

see the subsequent work of Barak et al. [3] for an exploration of this property and its role in obfuscation (we note that their positive results do not apply to the family of conjunctions).

As an example of the applications of a conjunction obfuscator, consider the following setting. There are k passwords, each controlling access to a particular type of resource. Each individual knows some subset of the k passwords, which corresponds to the resources it is allowed to access. A gatekeeper wishes to check whether an individual has access to some combination of resources, i.e., whether the individual knows a particular subset $S \subset [k]$ of the passwords, without revealing to an observer which combination it is checking. A conjunction, which takes as input k concatenated passwords, can check whether the passwords for resources in S are correct, while ignoring passwords for resources not in S. An obfuscation of this conjunction can be made public, and used to check whether an individual has access to that combination of resources without revealing which resources are being checked (nor, of course, what any of the passwords are) to any user that is denied access (A user with access can of course learn which of her passwords were in fact required).

1.1. *Our Construction and Its Security*

The main tool in our construction is multilinear maps. In particular, we utilize a recent candidate for graded encoding (a generalization of multilinear maps) due to [21].[1] We prove the security of our obfuscator when the conjunction is chosen from a distribution with sufficient entropy: namely when sampling $C = (W, V)$ from the distribution, even given the wildcard locations W, there is sufficient (superlogarithmic) entropy in V. We stress that this *does not* imply that the attacker is allowed to learn W; on the contrary, we prove that if C is drawn from a distribution with the aforementioned property, the adversary cannot learn anything, wildcard locations included.[2] As noted above, here we follow several works [10,16,26,27] which relax the security requirement to hold only when the circuit to be obfuscated is drawn from a distribution from a certain class (usually one with sufficient entropy).

We prove the above under two security assumptions on graded encodings schemes: The first is a translation of the SXDH assumption on bilinear groups to the setting of graded encoding schemes.[3] The second assumption is reminiscent of "Canetti's Assumption" [10] on Diffie–Hellman groups, which was introduced for the purpose of obfuscating point functions.

We conjecture that the construction is secure for *every* conjunction, but we were unable to produce a proof based on a well-established assumption (naturally, one can always take the security of the obfuscator as an assumption). As supportive evidence for the

[1] We use the *asymmetric* variant of the encoding scheme, where there are several distinct "source groups."

[2] We remark that in this case nothing at all can be learned from black box access to the function since it is infeasible to find an accepting input. We also remark that, for example, the conjunctions used for the k-resource application above naturally satisfy this condition, because of the entropy in each password.

[3] This assumption is actually known to be *false* for the construction and formulation of [21]. However, we show a more careful definition of the scheme and the assumption for which no attack is known. Also, no attack is known for the recent construction of Coron et al. [14].

conjectured security, we prove that the obfuscator is secure against *generic adversaries*: Ones that only use the group structure and not the representation of the group elements. This is similar to the generic group model of [30,33]. The proof of security against generic adversaries is non-trivial, and we view this as one of our main technical contributions. In fact, in subsequent work [8] we leverage the framework introduced in this work and provide the first generic proof of security for an obfuscator for general circuits.

We note that previous works on obfuscation [17,29] have also used the random oracle and generic group models to provide evidence for the security of constructions. However, subsequent works showed that obfuscation for general circuits is possible in the generic model despite being impossible in the standard model (see Sect. 1.2). Thus, one should not take generic model proofs to imply security (or even a possibility of security) in the standard model. Still, no impossibility is known for obfuscating simple classes such as conjunctions and we do view our generic model security proof as evidence that our construction is secure against simple (algebraic) attacks.

We proceed with an overview of our construction and results. As we explained, the obfuscator uses the recent construction of multi-linear maps via graded encoding schemes [21]. We begin with a high-level overview on the properties of multilinear maps that will be used. We then proceed with an overview of our construction, and state our two main results.

Multilinear Maps and Graded Encoding Schemes: Background We begin by recalling the notion of multilinear maps, due to Boneh and Silverberg [9]. Rothblum [31] considered the asymmetric case, where the groups may be different (this is crucial for our construction).

Definition 1.1. (*Asymmetric Multilinear Map* [9,31]) For $\tau+1$ cyclic groups $G_1,\ldots,G_\tau$, G_T of the same order p, a τ-multilinear map $e : G_1 \times \cdots \times G_\tau \to G_T$ has the following properties:

1. For elements $\{g_i \in G_i\}_{i=1,\ldots,\tau}$, index $i \in [\tau]$ and integer $\alpha \in \mathbb{Z}_p$, it holds that:

$$e(g_1,\ldots,\alpha\cdot g_i,\ldots,g_\tau) = \alpha \cdot g(g_1,\ldots,g_\tau)$$

2. The map e is non-degenerate: when its inputs are all generators of their respective groups $\{G_i\}$, then its output is a generator of the target group G_T.

Recently, Garg et al. [21] suggested a candidate for graded encoding, a generalization of (symmetric or asymmetric) multilinear maps. See Sect. 2.2 for a more complete overview of these objects. For this introduction, we treat them as a generalization of asymmetric multilinear maps in the following way. For a τ-multilinear map e, for the group G_i of prime order p, we consider the ring $\mathbb{Z}_p$. For an element $\sigma \in \mathbb{Z}_p$, we can think of g_i^σ as an "encoding" of σ in G_i. We denote this by $\mathsf{enc}_i(\sigma)$. We note that this encoding is easy to compute, but (presumably) hard to invert. The multilinear map e lets us take τ encodings $\{\mathsf{enc}_i(\sigma_i)\}_{i\in[\tau],\sigma_i\in\mathbb{Z}_p}$, and compute the target group encoding $\mathsf{enc}_T(\prod_i \sigma_i)$. Graded encoding schemes afford a similar functionality, albeit with randomized and noisy encodings, and with a procedure for testing equality of encoded elements in the target group.

Our Construction For a conjunction $C = (W, V)$ on n-bits inputs, the obfuscator uses the graded encoding scheme to obtain the above generalization to an $(n+1)$-multilinear map. For each input entry $i \in [n]$, the obfuscator picks ring elements $(\rho_{i,0}, \rho_{i,1}, \alpha_{i,0}, \alpha_{i,1})$ distributed as follows: if $i \notin W$, namely the entry isn't a wildcard, then the ring elements are independent and uniformly random. If $i \in W$, namely the entry is a wildcard, then the ring elements are uniformly random under the constraint that $\alpha_{i,0} = \alpha_{i,1}$. After picking the ring elements, the obfuscator outputs two pairs of encodings for each $i \in [n]$:

$$\{(w_{i,b} = \mathsf{enc}_i(\rho_{i,b}), \quad u_{i,b} = \mathsf{enc}_i(\rho_{i,b} \cdot \alpha_{i,b}))\}_{i\in[n],b\in\{0,1\}}$$

Note that if $i \in W$, then the ratio between the ring elements encoded in $u_{i,0}$ and $w_{i,0}$, is equal to the ratio between the ring elements encoded in $u_{i,1}$ and $w_{i,1}$ (these ratios are, respectively, $\alpha_{i,0}$ and $\alpha_{i,1}$, which are equal when $i \in W$). We remark that this part of the obfuscation depends only on the wildcards W, but not on the values V.

To complete the obfuscation, the obfuscator picks independent and uniformly random ring element ρ_{n+1}, and outputs a pair of encodings:

$$(w_{n+1} = \mathsf{enc}_{n+1}(\rho_{n+1}), u_{n+1} = \mathsf{enc}_{n+1}(\rho_{n+1} \cdot \underbrace{\prod_{i\in[n]} \alpha_{i,V[i]}}_{=\alpha_{n+1}}))$$

To evaluate the obfuscated program on an input $\vec{x} \in \{0,1\}^n$, we test equality between two multilinear products[4]:

$$e(\ldots, u_{i,\vec{x}[i]}, \ldots, w_{n+1}) \stackrel{?}{=} e(\ldots, w_{i,\vec{x}[i]}, \ldots, u_{n+1}) \tag{1}$$

The full construction is in Sect. 3.

Correctness Examining the two multilinear products in Eq. (1), the element encoded in the left-hand side is $(\prod_{i\in[n]} \rho_{i,\vec{x}[i]} \cdot \alpha_{i,\vec{x}[i]}) \cdot \rho_{n+1}$. The element encoded in the right-hand side is $(\prod_{i\in[n]} \rho_{i,\vec{x}[i]} \cdot \alpha_{i,V[i]}) \cdot \rho_{n+1}$. Thus, Eq. (1) holds if and only if:

$$\prod_{i\in[n]} \alpha_{i,\vec{x}[i]} = \prod_{i\in[n]} \alpha_{i,V[i]} \tag{2}$$

For $i \in W$ we have $\alpha_{i,0} = \alpha_{i,1}$, the contributions from the i-th group to both products in Eq. (2) are identical. For $i \notin W$, the contribution from the i-th group in the left-hand side of Eq. (2) is $\alpha_{i,\vec{x}[i]}$. In the right-hand side, the contribution is $\alpha_{i,V[i]}$. Except for a negligible probability of error, Eq. (2) holds if and only if all these contributions are identical, i.e., if and only if $\forall i \notin W : \vec{x}[i] = V[i]$.

[4]For the candidate of [21], the encodings are randomized, but there is a procedure for testing equality between encoded elements in the target group.

Security Security is not as straightforward. A slightly misleading intuition for security is that if a DDH-like assumption holds within each group G_i separately, then no observer can distinguish from that group's encodings whether $\alpha_{i,0} = \alpha_{i,1}$. This is true for each group in isolation, but it is insufficient because the obfuscation also includes encodings, in group G_{n+1}, of items that are correlated with the items encoded in group i. The multilinear map e might allow an adversary to distinguish whether the i-th entry is a wildcard.

For example, if in C all the entries are wildcards, the adversary can pick a random input, run the obfuscation, see that it accepts, and then by flipping the input bits one-by-one it can determine that all of the entries are wildcards. This attack clearly demonstrates that (for some conjunctions) an adversary can determine which entries are wildcards and which aren't. Note, however, that (for the specific example of a conjunction that is all-wildcards) this could also be accomplished using black box access to the conjunction.

Indeed, we prove the security of the obfuscator when the conjunction is drawn from a distribution, under mild conditions on the distribution's entropy. We conjecture that the obfuscator is actually secure for *any* conjunction, and as supporting evidence we show that it is secure against generic adversaries. An overview on both results follows.

Security for High Entropy We prove the security of our scheme in the case where $C = (W, V)$ is drawn from a distribution where the entropy of V given W is superlogarithmic. We do so by resurrecting the flawed argument described above: We use the entropy to remove the dependence between the elements in G_{n+1} and those in the other groups, and then apply DDH in each group.

We start by noting that this dependence is due to the relation

$$\alpha_{n+1} = \prod_{i \in [n]} \alpha_{i,V[i]}, \tag{3}$$

and if we could replace α_{n+1} with a completely uniform variable, independent of the other α's, we'd be done. To this end, we notice that Eq. (3) describes an (almost) pairwise independent hash function, whose seed are the values $\alpha_{i,b}$ and whose input is V. We show that such a hash function is a good entropy condenser, so that almost all of the entropy in V is preserved in α_{n+1}. (It is important to notice that the distinguisher has side information which depends on W, and therefore we must require that the *conditional entropy* is high.)

Once we establish that α_{n+1} has superlogarithmic entropy, we use a "Canetti-like Assumption" [10]: we assume a high-entropy element in the exponent of a random group generator is indistinguishable from uniform.[5] We thus isolate α_{n+1} from the dependence on the other α's, which allows us to apply DDH in groups $G_1, \ldots, G_n$, and obtain the final result: that the obfuscated program comes from a distribution that can be efficiently simulated.

The security proof is in Sect. 4.

[5] Wee [35] showed that these types of assumptions (hardness given only super-logarithmic entropy) are essential even for obfuscating point functions.

Security in The Generic Model We prove security against *generic adversaries*. A generic adversary is one that succeeds regardless of the representation of the encoding scheme. This is modeled by allowing it to only manipulate encodings in the graded encoding scheme via oracle access to an oracle for the operations that are available using the *evparams* parameters. We show that for any generic adversary $\mathcal{A}$, which takes as input an obfuscation and outputs a single bit, there exists a generic simulator $\mathcal{S}$ s.t. for any conjunction C, the adversary's output on an obfuscation of C is statistically close to the simulator's output given only black box access to C. The distribution of the adversary is taken over the choice of a *random* graded encoding scheme oracle: an oracle that represents each encoding in each group using a (long enough) uniformly random string.

In this model, since each element's encoding is uniformly random, and the obfuscation contains the encodings of distinct ring elements, the obfuscation of any conjunction is simply a collection of uniformly random strings. Thus simulating the obfuscator's output is easy. The main challenge is that *the outputs to oracle calls* on the string in the obfuscation are highly dependant on the conjunction C. It is thus not clear how the simulator can simulate the oracle's outputs. For example, each accepting input $\vec{x}$ for C specifies two possible inputs to the oracle implementing the multilinear map, which should both yield the same encoding in the target group. Indeed, simulating the oracle call outputs proves challenging. Moreover, the more generalized notion of graded encoding schemes permits more general generic operations.

The simulator $\mathcal{S}$ operates as follows. It feeds the adversary $\mathcal{A}$ with a "dummy obfuscation" containing uniformly random strings. It then follows $\mathcal{A}$'s calls to the graded encoding scheme (GES) oracle, and tries to simulate the output. For each call made by $\mathcal{A}$, we show how $\mathcal{S}$ can (efficiently) identify a polynomial size set X of inputs, such that if $\forall \vec{x} \in X, C(\vec{x}) = 0$, then the oracle's output is essentially independent of C and can be simulated. On the other hand, if there exists $\vec{x} \in X$ s.t. $C(\vec{x}) = 1$, then the simulator can use its black box access to C to identify this input. Once an accepting input is identified, the simulator can further use its block-box access to C to retrieve the conjunction's explicit description (W, V) (see Claim 3.3 below). Once the simulator knows (W, V) it can (perfectly) simulate the adversary's behavior. We view this proof of security for generic adversaries as one of our main technical contributions.

The full specification and treatment of the generic GES model, as well as the proof of security for generic adversaries, are in Sect. 5.

1.2. *Concurrent and Subsequent Work*

In subsequent work [7], we extend techniques from this paper and present an obfuscator for the more general class of d-CNF formulae, for any constant d. We prove security in the generic model. This proof builds on the proof in this paper, albeit with additional complications mostly stemming from the fact that an accepting input does not necessarily expose the obfuscated function.

Concurrently to this work and to [7], Garg et al. [22] use graded encoding schemes to present a candidate obfuscator for the class of all polynomial size circuits. They conjectured that their obfuscator is an *indistinguishability obfuscator* [6]. Based on the techniques from this work and from [7], we construct a variant of their obfuscator

and prove its security in the generic model [8] (no such proof is known for the [22] obfuscator). Subsequent works presented additional obfuscators with generic proofs.

Perhaps surprisingly, the generic proofs extended to the VBB setting as well. Since it is known that VBB obfuscation for general circuits is impossible, these results highlight the fact that one should not take generic model proofs as a guarantee for security in the standard model.

We emphasize that in this work we aim for VBB obfuscation of conjunctions (and provide a proof in the generic model). We also prove security in the standard model for high-entropy distributions on conjunctions (under hardness assumptions on graded encoding schemes, see above). While indistinguishability obfuscation has proved to be very valuable in the construction of many cryptographic primitives, it is not on its own known to imply a VBB obfuscator for conjunctions nor is it known to imply obfuscation for high-entropy families of conjunctions.

2. Preliminaries

Notation We use $\Delta(\cdot,\cdot)$ to indicate total variation distance (statistical distance). We use $\vec{1}$ (respectively, $\vec{0}$) to denote the all-1 (all-0) vector. We use $\mathbf{e}_i$ to denote the ith indicator vector. The dimension will be clear from the context in all of these cases.

Composite-Order Schwartz-Zippel We use the following corollary of the Schwartz-Zippel Lemma (the proof below is taken from [1]).

Corollary 2.1. *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. □

2.1. *Min-Entropy and Extraction*

The following are information-theoretic tools that will be required in our proof. The main notion of entropy used in this work is that of average min-entropy from [18], as well as its smooth version (see Definitions 2.2, 2.3 below). We then show that applying a pairwise independent hash function with a large enough image on an average min-entropy source, roughly preserves the average min-entropy (that is, it is an entropy condenser). This is derived from the generalized "crooked" leftover hash lemma [5,19].

We start by defining average min-entropy.

Definition 2.2. (*average min-entropy* [18]) Let X, Z be (possibly dependent) random variables, the average min-entropy of X conditioned on Z is:

$$\widetilde{\mathbf{H}}_\infty(X|Z) = -\log\left(\mathop{\mathbb{E}}_{z\leftarrow Z}\left[2^{-\mathbf{H}_\infty(X|Z=z)}\right]\right)$$

It follows from the definition that for every deterministic function f (that may depend on Z):

$$\widetilde{\mathbf{H}}_\infty(f(X)|Z) \leq \widetilde{\mathbf{H}}_\infty(X|Z). \tag{4}$$

We will also use a smooth variant introduced in [18, Appendix A] following [32].

Definition 2.3. (*smooth average min-entropy* [18]) Let X, Z be as above and let $\epsilon > 0$, then

$$\widetilde{\mathbf{H}}^\epsilon_\infty(X|Z) = \max_{(X',Z'):\Delta((X,Z),(X',Z'))\leq\epsilon} \widetilde{\mathbf{H}}_\infty(X'|Z').$$

For our next step, we will require the following tool.

Lemma 2.4. (generalized "crooked" LHL [5, Lemma 7.1]) *Let X, Z be random variables, let $\mathcal{H}$ be a pairwise independent hash family, and let $h \leftarrow \mathcal{H}$ be a properly sampled function from this family, finally let f be a function of image size K, then*

$$\Delta((f(h(X)), Z, h), (f(U), Z, h)) \leq \frac{1}{2}\sqrt{K} \cdot 2^{-\frac{1}{2}\widetilde{\mathbf{H}}_\infty(X|Z)},$$

where U is uniformly distributed in the domain of f.

We will show that pairwise independent function condense average min-entropy in the following way.

Lemma 2.5. *Let X, Z be random variables, let $\mathcal{H}$ be a pairwise independent hash family whose output is a binary string of length $\geq \left\lfloor \widetilde{\mathbf{H}}_\infty(X|Z) - 2\log(1/\epsilon) + 2 \right\rfloor$, for some $\epsilon > 0$. Then letting $h \leftarrow \mathcal{H}$ be a properly sampled function from this family, it holds that*

$$\widetilde{\mathbf{H}}^\epsilon_\infty(h(X)|Z, h) \geq \widetilde{\mathbf{H}}_\infty(X|Z) - 2\log(1/\epsilon) + 1.$$

Proof. Let X, Z, $\mathcal{H}$, h, ϵ be as in the lemma statement. Our goal is to show that there exists a random variable Y such that

$$\Delta((h(X), Z, h), (Y, Z, h)) \leq \epsilon,$$

and

$$\widetilde{\mathbf{H}}_\infty(Y|Z, h) \geq \widetilde{\mathbf{H}}_\infty(X|Z) - 2\log(1/\epsilon) + 1.$$

Let f be the function that outputs the first k bits of its input, for

$$k = \left\lfloor \widetilde{\mathbf{H}}_\infty(X|Z) - 2\log(1/\epsilon) + 2 \right\rfloor \geq \widetilde{\mathbf{H}}_\infty(X|Z) - 2\log(1/\epsilon) + 1,$$

and note that $f(U)$ is uniform over $\{0, 1\}^k$ (in fact, we can use any function that has this property).

Then by Lemma 2.4, it holds that

$$\Delta((f(h(X)), Z, h), (f(U), Z, h)) \leq \epsilon.$$

Now, consider a 2-step process for sampling the joint distribution $(h(X), Z, h)$: first, sample $(f(h(X)), Z, h)$ from the appropriate marginal distribution; and then sample $h(X)$ conditioned on the previously sampled values.

We define Y using the following process: First, sample a tuple according to the distribution $(f(U), Z, h)$, and then apply the second stage of the sampling process from above. The result will be the distribution (Y, Z, h). Clearly,

$$\Delta((h(X), Z, h), (Y, Z, h)) = \Delta((f(h(X)), Z, h), (\underbrace{f(Y)}_{=f(U)}, Z, h)) \leq \epsilon,$$

where the first equality is since there is a deterministic mapping (f) from the left-hand side to the right-hand side, and a randomized mapping (the second step sampler) from the right-hand side to the left-hand side.

To conclude, we notice that

$$\begin{aligned}\widetilde{\mathbf{H}}_\infty(Y|Z, h) &\geq \widetilde{\mathbf{H}}_\infty(f(Y)|Z, h) = \widetilde{\mathbf{H}}_\infty(\{0,1\}^k|Z, h) \\ &= k \geq \widetilde{\mathbf{H}}_\infty(X|Z) - 2\log(1/\epsilon) + 1.\end{aligned}$$

□

2.2. *Graded Encoding Schemes and Assumptions*

We begin with the definition of a graded encoding scheme, due to Garg et al. [21]. While their construction is very general, for our purposes a more restricted setting is sufficient as defined below.

Definition 2.6. (*τ-Graded Encoding Scheme* [21]) A τ-encoding scheme for a ring R is a collection of sets $\mathcal{S} = \{S_{\mathbf{v}}^{(\alpha)} \subset \{0,1\}^* : \mathbf{v} \in \{0,1\}^\tau, \alpha \in R\}$, with the following properties:

1. For every index $\mathbf{v} \in \{0,1\}^\tau$, the sets $\{S_{\mathbf{v}}^{(\alpha)} : \alpha \in R\}$ are disjoint, and so they are a partition of the indexed set $S_{\mathbf{v}} = \bigcup_{\alpha \in R} S_{\mathbf{v}}^{(\alpha)}$.
2. There are binary operations "+" and "−" such that for all $\mathbf{v} \in \{0,1\}^\tau$, $\alpha_1, \alpha_2 \in R$ and for all $u_1 \in S_{\mathbf{v}}^{(\alpha_1)}$, $u_2 \in S_{\mathbf{v}}^{(\alpha_2)}$:

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

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

holds that

$$u_1 \times u_2 \in S^{(\alpha_1 \cdot \alpha_2)}_{\mathbf{v}_1+\mathbf{v}_2},$$

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

Definition 2.7. (*Efficient Procedures for a τ-Graded Encoding Scheme* [21]) 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 $S^{(\alpha)}_{\vec{1}}$). In addition, the procedure outputs a subset *evparams* $\subset$ *params* that is sufficient for computing addition, multiplication and zero testing[6] (but possibly insufficient for encoding or for randomization).
- *Ring Sampler* $\mathsf{samp}(params)$ outputs a "level zero encoding" $a \in S^{(\alpha)}_{\vec{0}}$ for a nearly uniform $\alpha \in_R R$.
- *Encode and Re-Randomize*[7] $\mathsf{encRand}(params, i, a)$ takes as input an index $i \in [\tau]$ and $a \in S^{(\alpha)}_{\vec{0}}$, and outputs an encoding $u \in S^{(\alpha)}_{\mathbf{e}_i}$, where the distribution of u is (statistically close to being) only dependent on α and not otherwise dependent of a.
- *Addition and Negation* $\mathsf{add}(evparams, u_1, u_2)$ takes $u_1 \in S^{(\alpha_1)}_{\mathbf{v}}, u_2 \in S^{(\alpha_2)}_{\mathbf{v}}$, and outputs $w \in S^{(\alpha_1+\alpha_2)}_{\mathbf{v}}$. (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, $\mathsf{negate}(evparams, u_1) \in S^{(-\alpha_1)}_{\mathbf{v}}$.
- *Multiplication* $\mathsf{mult}(evparams, u_1, u_2)$ takes $u_1 \in S^{(\alpha_1)}_{\mathbf{v}_1}, u_2 \in S^{(\alpha_2)}_{\mathbf{v}_2}$. If $\mathbf{v}_1 + \mathbf{v}_2 \in \{0,1\}^\tau$ (i.e., every coordinate in $\mathbf{v}_1 + \mathbf{v}_2$ is at most 1), then mult outputs $w \in S^{(\alpha_1 \cdot \alpha_2)}_{\mathbf{v}_1+\mathbf{v}_2}$. 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 $u \in S^{(0)}_{\vec{1}}$, and 0 otherwise.

Remark 2.8. (*The Order of the Ring R.*) In this work, the ring R will always be (congruent to) $\mathbb{Z}_p$ for an integer p. We allow p to be either prime or a product of distinct primes. In general, we denote $p = \prod_{i=1}^{\sigma} p_i$ such that p_i are distinct primes. We require that all p_i's are super-polynomial in the security parameter. Indeed, some known candidate multilinear maps (including the recent [15]) use composite order p, and their security is severely compromised if p_i are chosen to be polynomial in the security parameter. For such fvalues of p, we notice that:

[6] The "zero testing" parameter **pzt** defined in [21] is a part of *evparams*.

[7] This functionality is not explicitly provided by Garg et al. [21]; however, it can be obtained by combining their encoding and re-randomization procedures.

1. For any polynomial $poly(\cdot)$:

$$\frac{poly(\lambda)^\sigma}{p} = \prod_{i=1}^{\sigma}\left(\frac{poly(\lambda)}{p_i}\right) = negl(\lambda).$$

2. Almost all elements in the ring are units, namely

$$\Pr_{x\leftarrow\mathbb{Z}_p}\left[x \notin \mathbb{Z}_p^*\right] = 1 - \prod_{i=1}^{\sigma}(1 - 1/p_i) = negl(\lambda).$$

Comparison with the Definition of [21]. We now explain the relation between our definitions of graded encoding and those appearing in [21]. This section is intended to those readers who are familiar with the [21] work and can safely be skipped by other readers as our work stands self-contained without it.

We start by noting that Definition 2.6 above is the special case of the [21] construction in which we consider only binary index vectors (in the [21] 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-text index $\mathbf{vzt} = \vec{1}$).

In the [14,21] 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.

It is also important to notice that our Definition 2.7 deviates from that of [21] as 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 [14,21], *evparams* only needs to contain the zero-test parameter **pzt** (as well as the global modulus).

2.2.1. *Hardness Assumptions*

In this work, we will use two hardness assumptions over graded encoding schemes. The first, which we call "graded external DDH" (or GXDH, Assumption 2.9 below) is an analog of the symmetric external DH assumption (SXDH), instantiated for the multilinear case. The second assumption (GCAN Assumption 2.10) is an analog of Canetti's assumption [10], that taking a random generator to a high-entropy power results in a random-looking element. We note that we make these assumptions against non-uniform adversaries. We further note that both assumptions hold in the generic model.

Assumption 2.9. (*Graded External DH*) Let $(params, evparams) \leftarrow \mathsf{InstGen}(1^\lambda, 1^\tau)$, for all $i = 1, \ldots, \tau$, sample $r_{i,0}, r_{i,1}, a_{i,0}, a_{i,1} \leftarrow \mathsf{samp}(params)$ and consider the following values:

$$w_{i,0} \leftarrow \mathsf{encRand}(params, i, r_{i,0}) \qquad w_{i,1} \leftarrow \mathsf{encRand}(params, i, r_{i,1})$$
$$u_{i,0} \leftarrow \mathsf{encRand}(params, i, r_{i,0} \times a_{i,0}) \qquad u_{i,1} \leftarrow \mathsf{encRand}(params, i, r_{i,1} \times a_{i,1})$$
$$u'_{i,1} \leftarrow \mathsf{encRand}(params, i, r_{i,1} \times a_{i,0})$$

The GXDH assumption is that for every choice of $\tau \in \mathbb{N}$ and $i^* \in [\tau]$, no ensemble of polynomial time adversaries can have non-negligible advantage in distinguishing the distributions:

$$\left(evparams, \{(w_{i,0}, u_{i,0}, w_{i,1}, u_{i,1}, u'_{i,1})\}_{i \neq i^*}, (w_{i^*,0}, u_{i^*,0}, w_{i^*,1}, u_{i^*,1})\right)$$
and
$$\left(evparams, \{(w_{i,0}, u_{i,0}, w_{i,1}, u_{i,1}, u'_{i,1})\}_{i \neq i^*}, (w_{i^*,0}, u_{i^*,0}, w_{i^*,1}, u'_{i^*,1})\right)$$

We note that a stronger version of this assumption, where the distinguisher is given access to *params* rather than *evparams*, was presented in the early versions of [21]. It was later shown that this stronger assumption is false, see later versions of [21] for the attack. This later led to a number of attacks on [14,21] known as "zeroizing attacks," columinating in a complete break of [14] and nearly complete break of [21] in the case where *params* are given to the attacker [13,25]. In some cases, this attack can be launched also when only *evparams* are given (in addition to other encodings) [12]. A new candidate by Coron, Lepoint and Tibouchi [15] is currently not known to be affected by zeroizing attacks.

Since we only provide our distinguisher with *evparams*, it may not be able to generate DDH tuples by itself. We therefore provide it with correctly labeled DDH samples for all groups except i^*. This is the minimal assumption that is required for our construction; however, we conjecture that a stronger variant where the adversary is allowed to receive an unbounded number of labeled samples at any group (including i^*) is also true.

For our next assumption, we introduce the following notation. Consider a distribution D over $S_{\mathbf{v}} = \cup_{\alpha \in R} S_{\mathbf{v}}^{(\alpha)}$. The distribution $\mathsf{enc}^{-1}(D)$ is defined by the following process: Sample $x \leftarrow D$, let α be such that $x \in S_{\mathbf{v}}^{(\alpha)}$, output α. We also recall the definition of smooth average min-entropy (see Definition 2.3 above).

Assumption 2.10. (*"Graded Canetti"*) Let $(params, \mathbf{pzt}) \leftarrow \mathsf{InstGen}(1^\lambda, 1^\tau)$ and let $\{(D_\lambda, Z_\lambda)\}_{\lambda \in \mathbb{N}}$ be a distribution ensemble over $S_{\vec{0}} \times \{0,1\}^*$, such that

$$\widetilde{\mathbf{H}}_\infty^\epsilon(\mathsf{enc}^{-1}(D_\lambda) | Z_\lambda) \geq h(\lambda),$$

for some $\epsilon = negl(\lambda)$ and function $h(\lambda) = \omega(\log \lambda)$.

The GCAN assumption is that no ensemble of polynomial time adversaries and indices i can have non-negligible advantage in distinguishing the distributions

$$(params, evparams, w, u, z) \text{ and } (params, evparams, w, u', z),$$

where $(params, evparams) \leftarrow \mathsf{InstGen}(1^\lambda, 1^\tau)$, $r \leftarrow \mathsf{samp}(params)$, $w \leftarrow \mathsf{encRand}(params, i, r)$, $(x, z) \leftarrow (D_\lambda, Z_\lambda)$, $u \leftarrow \mathsf{encRand}(params, i, r \times x)$, $u' \leftarrow \mathsf{encRand}(params, i, \mathsf{samp}(params))$. (Note that in this definition, the distinguisher is given both *params* and *evparams*.)

This assumption is consistent with our knowledge on candidate graded encoding schemes. However, if we want to make an even weaker assumption, we can set the minimal entropy requirement to be higher than just $\omega(\log \lambda)$. The constructions in this paper can trivially be adapted to such weaker variants (with the expected degradation in security).

2.3. *Obfuscation*

Definition 2.11. (*Virtual Black Box Obfuscator* [6]) 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 an *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 $negl(\lambda)$ 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* $C \in \mathcal{C}_n$:

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

Remark 2.12. 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 2.13. (*Average-Case Secure Virtual Black Box*) Let $\mathcal{C} = \{\mathcal{C}_n\}_{n\in\mathbb{N}}$ be a family of circuits and $\mathcal{O}$ a PPTM as in Definition 2.11. Let $\mathcal{D} = \{\mathcal{D}_n\}_{n\in\mathbb{N}}$ be an ensemble of distribution families $\mathcal{D}_n$, where each $D \in \mathcal{D}_n$ is a distribution over $\mathcal{C}_n$.

$\mathcal{O}$ is an *obfuscator* for the distribution class $\mathcal{D}$ over the circuit family $\mathcal{C}$, if it satisfies the functionality and polynomial slowdown properties of Definition 2.11 with respect to $\mathcal{C}$, but the virtual black box property is replaced with:

3. *Distributional 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 $n \in \mathbb{N}$, every *distribution* $D \in \mathcal{D}_n$ (a distribution over $\mathcal{C}_n$), and every predicate $P : C_n \to \{0,1\}$:

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

Remark 2.14. Our proof of average-case security for the conjunction obfuscator (Theorem 4.2) is in fact stronger. We show a simulator $\mathcal{S}$ that does not even require black box access to the circuit C. Rather, for a circuit C drawn from a distribution in $\mathcal{D}$, the probability of predicting $P(C)$ from an obfuscation of C, is the same as the probability of predicting $P(C)$ from a "dummy obfuscation" that is independent of C. See the proof for further details.

3. Obfuscating Conjunctions

In this section, we present our obfuscator for conjunctions ConjObf (Fig. 1). We provide a proof of security for functions that are not determined by the locations of the wildcards in Sect. 4. Finally, in Sect. 5 we provide evidence of the security of our construction for *any* conjunction, by proving that it is secure against generic adversaries that do not use the representation of the specific graded encoding scheme.

We start with a formal definition of the class of conjunctions.

Definition 3.1. (*n-bit Conjunction*) For an input length n, a conjunction $C = (W, V) : \{0,1\}^n \to \{0,1\}$ is a predicate on n-bit inputs, which is defined by two vectors $W, V \in \{0,1\}^n$. For every input $\vec{x} \in \{0,1\}^n, C(\vec{x}) = 1$ iff for all $i \in [n], W[i] = 1$ or $V[i] = \vec{x}[i]$. For the sake of unity of representation, we require that whenever $W[i] = 1$, it holds that $V[i] = 0$.

We often alternate between treating W as an index vector and treating it as a subset of $[n]$. If $i \in W$ then we say that i is a *wildcard location*.

An ensemble of conjunctions is defined in the standard way.

Definition 3.2. (*Conjunction Ensemble*) A conjunction ensemble $\mathcal{C} = \{C_n\}_{n \in \mathbb{N}}$ is a collection of conjunctions $C_n : \{0,1\}^n \to \{0,1\}$, one for each input length.

We now state a useful (for our purposes) property of conjunctions:

Claim 3.3. *There exists an efficient algorithm $\mathcal{B}$, such that for any conjunction $C = (W, V)$, and any accepting input $\vec{x}$ of C, $\mathcal{B}$ can recover (W, V):*

$$\forall C = (W, V), \forall \vec{x} : C(\vec{x}) = 1, \quad \mathcal{B}^C(\vec{x}) = (W, V)$$

Obfuscator ConjObf**, on input** $(1^\lambda, 1^n, C = (W, V))$

1. generate $(params, evparams) \leftarrow \mathsf{InstGen}(1^\lambda, 1^{n+1})$
2. for $i \in [n]$:
   if $i \in W$, then: $a_{i,0} = a_{i,1} \leftarrow \mathsf{samp}(params) \in S_0^{(\alpha_{i,0})} = S_0^{(\alpha_{i,1})}$
   if $i \notin W$, then: $a_{i,0} \leftarrow \mathsf{samp}(params) \in S_0^{(\alpha_{i,0})}$, $a_{i,1} \leftarrow \mathsf{samp}(params) \in S_0^{(\alpha_{i,1})}$
3. for $i \in [n]$:
   $r_{i,0} \leftarrow \mathsf{samp}(params) \in S_0^{(\rho_{i,0})}, r_{i,1} \leftarrow \mathsf{samp}(params) \in S_0^{(\rho_{i,1})}$
   $s_{i,0} \leftarrow r_{i,0} \times a_{i,0} \in S_0^{(\rho_{i,0} \cdot \alpha_{i,0})}$, $s_{i,1} \leftarrow r_{i,1} \times a_{i,1} \in S_0^{(\rho_{i,1} \cdot \alpha_{i,1})}$
   $w_{i,0} \leftarrow \mathsf{encRand}(params, i, r_{i,0}) \in S_{\mathbf{e}_i}^{(\rho_{i,0})}$, $w_{i,1} \leftarrow \mathsf{encRand}(params, i, r_{i,1}) \in S_{\mathbf{e}_i}^{(\rho_{i,1})}$
   $u_{i,0} \leftarrow \mathsf{encRand}(params, i, s_{i,0}) \in S_{\mathbf{e}_i}^{(\rho_{i,0} \cdot \alpha_{i,0})}$, $u_{i,1} \leftarrow \mathsf{encRand}(params, i, s_{i,1}) \in S_{\mathbf{e}_i}^{(\rho_{i,1} \cdot \alpha_{i,1})}$
4. $a_{n+1} \leftarrow \left(\prod_{i \in [n]} a_{i,V[i]}\right) \in S_0^{(\prod_{i \in [n]} \alpha_{i,V[i]})}$
   $r_{n+1} \leftarrow \mathsf{samp}(params) \in S_0^{(\rho_{n+1})}$
   $s_{n+1} \leftarrow r_{n+1} \times a_{n+1} \in S_0^{(\rho_{n+1} \cdot \prod_{i \in [n]} \alpha_{i,V[i]})}$
   $w_{n+1} \leftarrow \mathsf{encRand}(params, n+1, r_{n+1}) \in S_{\mathbf{e}_{n+1}}^{(\rho_{n+1})}$
   $u_{n+1} \leftarrow \mathsf{encRand}(params, n+1, s_{n+1}) \in S_{\mathbf{e}_{n+1}}^{(\rho_{n+1} \cdot \prod_{i \in [n]} \alpha_{i,V[i]})}$
5. output the obfuscation: $(evparams, \{(w_{i,0}, u_{i,0}), (w_{i,1}, u_{i,1})\}_{i \in [n]}, (w_{n+1}, u_{n+1}))$

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

1. $\mathbf{t} \leftarrow \left(w_{n+1} \times \prod_{i \in [n]} u_{i,\vec{x}[i]}\right) \in S_{(1,1,\ldots,1)}^{\left(\rho_{n+1} \cdot (\prod_{i \in [n]} \rho_{i,\vec{x}[i]} \cdot \alpha_{i,\vec{x}[i]})\right)}$
2. $\mathbf{t}' \leftarrow \left(u_{n+1} \times \prod_{i \in [n]} w_i\right) \in S_{(1,1,\ldots,1)}^{\left(\rho_{n+1} \cdot (\prod_{i \in [n]} \alpha_{i,V[i]}) \cdot (\prod_{i \in [n]} \rho_{i,\vec{x}[i]})\right)}$
3. output the bit: $\mathsf{isZero}(evparams, (\mathbf{t} - \mathbf{t}'))$.

Fig. 1. Obfuscator for Conjunctions.

Proof. Take $n = |\vec{x}|$, the algorithm $\mathcal{B}$ enumerates over the bits of $\vec{x}$. For each bit i, it flips the i-th bit of $\vec{x}$: $\vec{x}^{(i)} = \vec{x} \oplus \mathbf{e}_i$, and checks whether $C(\vec{x}^{(i)}) = 1$. If so, then i must be a wildcard: $W[i] = 1$ and $V[i] = 0$. Otherwise, i is not a wildcard: $W[i] = 0$ and $V[i] = \vec{x}[i]$. □

Our obfuscator for the class of conjunctions is presented in Fig. 1. Correctness follows in a straightforward manner as described in the following lemma. We note that the error is one sided, it is always the case that if $C(\vec{x}) = 1$ then for the obfuscated program $\mathcal{O}_C(\vec{x}) = 1$ as well.

Lemma 3.4. (Obfuscator Functionality) *Let C be an n-variable conjunction and consider its obfuscation $\mathcal{O}_C = \mathsf{ConjObf}(C, 1^n, 1^\lambda)$. Then for all $\vec{x}$,*

$$\Pr[\mathcal{O}_C(\vec{x}) \neq C(\vec{x})] \leq poly(n)/p,$$

where $p = 2^{\Omega(\lambda)}$ is the order of the group in the graded encoding scheme, and the probability is taken over the randomness of ConjObf.

Proof. Let $C, \mathcal{O}_C$ be as in the lemma statement. Let $\vec{x}$ be an input and let us examine $\mathcal{O}_C(\vec{x})$ first for the case where $C(\vec{x}) = 1$ and then the case where $C(\vec{x}) = 0$.

If $C(\vec{x}) = 1$ then by the definition of ConjObf

$$\rho_{n+1} \cdot (\Pi_{i\in[n]}\rho_{i,\vec{x}[i]} \cdot \alpha_{i,\vec{x}[i]}) = \rho_{n+1} \cdot (\Pi_{i\in[n]}\alpha_{i,V[i]}) \cdot (\Pi_{i\in[n]}\rho_{i,\vec{x}[i]}),$$

and therefore $\mathcal{O}_C(\vec{x}) = 1$.

Otherwise, consider the polynomial over $\mathbb{Z}_p$

$$p(\vec{\rho}, \vec{\alpha}) = \rho_{n+1} \cdot (\Pi_{i\in[n]}\rho_{i,\vec{x}[i]} \cdot \alpha_{i,\vec{x}[i]}) - \rho_{n+1} \cdot (\Pi_{i\in[n]}\alpha_{i,V[i]}) \cdot (\Pi_{i\in[n]}\rho_{i,\vec{x}[i]}).$$

This polynomial is not identically zero and it is defined over $poly(n)$ uniformly distributed variables. Therefore, since p has no small divisors (Remark 2.8), then by Schwartz-Zippel (Corollary 2.1) it will be zero with probability at most $poly(n)^\sigma / p = negl(\lambda)$. □

As a concluding remark, we note that if our graded encoding scheme has the property that $p \gg 2^n$ (which is indeed achievable in the candidate of [21]), then a stronger correctness guarantee, as mentioned in Remark 2.12, can be achieved by using the union bound. In this parameter range, the proof of security also becomes somewhat simpler (see Sect. 4). However, we want to present our scheme in the most generic way so as to be compatible with possible choices of the security parameter and with future graded encoding schemes.

4. Security from GXDH and GCAN

In this section, we prove that ConjObf is a secure distributional black box obfuscator for any distribution over the conjunctions family for which the function is hard to determine (i.e., has super-logarithmic entropy) even if the locations of all the wildcards are known. Namely, there is sufficient min-entropy in V even given W (recall that $V[i] = 0$ wherever $W[i] = 1$).

Definition 4.1. (*equivocality given wildcards*) Let $\mathcal{C}$ be the class of conjunctions, and let $\mathcal{D} = \{\mathcal{D}_\lambda\}$ be an ensemble of families of distributions. We say that $\mathcal{D}$ is *equivocal given the wildcards* if there exists $h(\lambda) = \omega(\log \lambda)$ such that for all $D \in \mathcal{D}_\lambda$, if $(V, W) \leftarrow D$ then

$$\widetilde{\mathbf{H}}_\infty(V|W) \geq h(\lambda).$$

We will prove the security of ConjObf for such functions under the GXDH and GCAN assumptions (see Sect. 2.2).

Theorem 4.2. *Based on the GXDH and GCAN assumptions, the algorithm* ConjObf *is an average-case black box obfuscator for ensembles of distribution families that are equivocal given the wildcards.*

Proof. We start by stating a claim that will be used later on in the proof.

Claim 4.3. *Let p be an integer and let $k \in \mathbb{N}$ be integer. Consider the hash family $\mathcal{H} \subseteq \{0,1\}^k \to \mathbb{Z}_p^*$, where each function in $\mathcal{H}$ is defined by a sequence $a_0, a_1, \ldots, a_k \in \mathbb{Z}_p^*$ and*

$$\mathcal{H}_{a_0,a_1,\ldots,a_k}(x_1,\ldots,x_k) = a_0 \cdot \prod_{i\in[k]} a_i^{x_i},$$

then $\mathcal{H}$ is pairwise independent.

This claim follows in a straightforward manner since $\mathcal{H}$ (defined therein) is a random linear function "in the exponent."

Consider a function $C = (W, V)$ drawn from a distribution $\mathcal{D}_\lambda$, and consider the distribution of a properly obfuscated program $\mathsf{ConjObf}(C)$. We will show, using a sequence of hybrids, that this distribution is computationally indistinguishable from one that does not depend on C, even for a distinguisher who knows the value of the predicate $P(C)$. This will immediately imply a simulator. We note that our proof works even for $P(C)$ with multiple-bit output, so long as $h(\lambda) - |P(C)| = \omega(\log\lambda)$.

1. In this hybrid, we use $\mathsf{ConjObf}$ as prescribed:

$$\mathcal{O}_C = \mathsf{ConjObf}(C) = \left(params, \mathbf{pzt}, \left\{(w_{i,b}, u_{i,b})\right\}_{i\in[n],b\in\{0,1\}}, (w_{n+1}, u_{n+1})\right).$$

2. We change step 4 of the obfuscator, and sample $a_{n+1} \leftarrow \mathsf{samp}(params)$.
   We show that the resulting $\mathcal{O}_C$ distribution is computationally indistinguishable from the previous hybrid under the GCAN assumption (Assumption 2.10), even when the distinguisher knows $P(C)$. Namely, we will show that for some negligible ϵ, the distributions in the previous hybrid are such that

$$\widetilde{\mathbf{H}}^{\epsilon}_{\infty}(\alpha_{n+1} | \left\{(w_{i,b}, u_{i,b})\right\}_{i\in[n],b\in\{0,1\}}, P(C)) = \omega(\log\lambda), \tag{5}$$

   which will allow us to apply GCAN and conclude that α_{n+1} can be replaced by a uniform variable.
   To show that Eq. (5) holds, we present a slightly different way to generate the variables $\alpha_{i,b}$ (note that from this point and on, we are a completely information-theoretic setting, so we will not worry about computational aspects). We will first sample $\{\hat{\alpha}_{i,b}\}_{i\in[n],b\in\{0,1\}}$ completely uniformly in $\mathbb{Z}_p$, and then set $\alpha_{i,b}$ as follows. If $W[i] = 0$ then $\alpha_{i,0} = \hat{\alpha}_{i,0}, \alpha_{i,1} = \hat{\alpha}_{i,1}$; and if $W[i] = 1$ then $\alpha_{i,0} = \alpha_{i,1} = \hat{\alpha}_{i,0}$. Note that the resulting distribution of the α's is exactly as prescribed. We let E denote the event where all $\hat{\alpha}_{i,b} \in \mathbb{Z}_p^*$. As noted in Remark 2.8, this event happens with all but negligible probability: $\Pr[\mathsf{E}] = 1 - negl(\lambda)$.
   We notice that conditioned on E, $\hat{\alpha}_{i,b}$ are invertible, and therefore we can write

$$\alpha_{n+1} = \prod_{i\in[n]} \alpha_{i,0}^{1-V[i]}\alpha_{i,1}^{V[i]} = \prod_{i\in[n]} \hat{\alpha}_{i,0}^{1-V[i]}\hat{\alpha}_{i,1}^{V[i]} = \prod_{i\in[n]} \hat{\alpha}_{i,0} \cdot \prod_{i\in[n]} (\hat{\alpha}_{i,1}/\hat{\alpha}_{i,0})^{V[i]} \tag{6}$$

where the second equality is since α and $\hat{\alpha}$ only differ where $W[i] = V[i] = 0$. By Claim 4.3 it follows, therefore, that α_{n+1} is the output of a pairwise independent hash function applied to V.
We proceed to apply Lemma 2.5. We note that $\widetilde{\mathbf{H}}_\infty(V|W, P(C)) \geq \widetilde{\mathbf{H}}_\infty(V|W) - |P(C)| \geq h(\lambda) - 1$. Therefore, there must exist $h'(\lambda) = \omega(\log \lambda)$ such that $h'(\lambda) \leq h(\lambda) - 1$, and in addition the length of α_{n+1} is at least $h'(\lambda)/3 + 2$. We can thus apply Lemma 2.5 with $\epsilon' = 2^{-h'(\lambda)/3} = negl(\lambda)$ to argue that

$$\widetilde{\mathbf{H}}^{\epsilon'}_\infty \left(\alpha_{n+1}|W, P(C), \left(\{\hat{\alpha}_{i,b}\}|\mathsf{E}\right)\right) \geq h'(\lambda)/3 = \omega(\log \lambda). \tag{7}$$

Letting $\epsilon = \epsilon' + \Pr[\mathsf{E}]$, this implies that

$$\widetilde{\mathbf{H}}^{\epsilon}_\infty \left(\alpha_{n+1}|W, P(C), \{\hat{\alpha}_{i,b}\}\right) \geq \widetilde{\mathbf{H}}^{\epsilon'}_\infty \left(\alpha_{n+1}|W, P(C), \left(\{\hat{\alpha}_{i,b}\}|\mathsf{E}\right)\right) \geq h'(\lambda)/3 = \omega(\log \lambda). \tag{8}$$

Finally, Eq. (5) follows by noticing that there is an invertible mapping between W, $\{\hat{\alpha}_{i,b}\}$ and $\left\{(w_{i,b}, u_{i,b})\right\}_{i\in[n],b\in\{0,1\}}$.
It is interesting to note that this hybrid (and therefore our entire argument) works not only for predicates. In fact, ℓ-bit functions of the circuit C can be used, so long as $h(\lambda) - \ell = \omega(\log \lambda)$.
At this point, $\mathcal{O}_C$ does not depend on V anymore, however it still depends on W via step 2 of ConjObf.

3. We change step 2 of the obfuscator to always act as if $i \notin W$, namely $\alpha_{i,0}$ and $\alpha_{i,1}$ are uniform and independent.
A sequence of n hybrids will show that any adversary distinguishing this distribution from the previous one, can be used to break GXDH with only a factor n loss in the advantage. This implies that the hybrids are computationally indistinguishable assuming GXDH. Note that knowledge of $P(C)$ (or even of C in its entirety) is useless for the distinguisher at this point.

After the last hybrid, we are at a case where all $a_{i,b}$, $r_{i,b}$, a_{n+1}, r_{n+1} are completely independent of each other, and are sampled in the same way regardless of (V, W). It follows that our final distribution is independent of C, but produces $\mathcal{O}_C$ indistinguishable from ConjObf(C) (even given $P(C)$). Since this distribution is efficiently sampleable (via the process we describe in the proof), the theorem follows. □

5. Security in the Generic Graded Encoding Scheme (GES) Model

5.1. *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, analogous to the *generic group model* (see Shoup [33] and Maurer [30]). 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 2.7. 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 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 $\mathbf{v} \in \{0,1\}^\tau$, the elements of the indexed set $S_\mathbf{v} = \bigcup_{\alpha \in R} S_\mathbf{v}^{(\alpha)}$ 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 $s_1 \in S_\mathbf{v}^{(\alpha_1)}$, $s_2 \in S_\mathbf{v}^{(\alpha_2)}$, encoding indexed ring elements $(\mathbf{v}, \alpha_1)$, $(\mathbf{v}, \alpha_2)$ (respectively), and obtaining a string $s \in S_\mathbf{v}^{(\alpha_1+\alpha_2)}$, encoding the indexed ring element $(\mathbf{v}, (\alpha_1 + \alpha_2))$.

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

Remark 5.1. (*Unique versus Randomized Representation*) We note that the known candidate of secure graded encoding schemes [21] *does 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.

We remark 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. 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 $\mathbf{v} \in \{0,1\}^\tau$ and ring element $\sigma \in R$), its encoding is of length $\ell = (\tau \cdot \log|R| \cdot poly(\lambda))$. 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 oracle $\mathcal{RG}$ maintains a table T (initially empty) containing the indexed ring elements that were provided as outputs to prior calls made by $\mathcal{A}$. As the execution of $\mathcal{A}$ proceeds, its oracle calls are handled one at a time. On each such call, the oracle $\mathcal{RG}$ checks whether every input argument that encodes an indexed ring element is the output of a previous call to $\mathcal{RG}$. If not, then the oracle immediately returns $\perp$ (at least one input argument is an invalid encoding). Otherwise, the oracle retrieves from its table T the indexed ring elements encoded in all input arguments. Using these, the oracle computes the correct output indexed ring element (or returns $\perp$ if the inputs are invalid). If this output already has an entry in the table T, then its encoding has already been determined, and can be provided as output. Otherwise, the oracle $\mathcal{RG}$ chooses a new uniformly random string in $\{0,1\}^{\ell}$, outputs this string as the output encoding, and adds the appropriate entry to the table T.

The definition of secure obfuscation in the random GES model is as follows.

Definition 5.2. (*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 2.11.

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 2.11 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 $C \in \mathcal{C}_n$:

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

We remark that while it makes sense to allow $\mathcal{S}$ to access the oracle $\mathcal{RG}$, this is in fact not necessary. This is since $\mathcal{RG}$ can be implemented in polynomial time (as described above), and therefore $\mathcal{S}$ can just implement it by itself.

5.2. *Security of* ConjObf

Theorem 5.3. (Security against Generic Adversaries) *The algorithm* ConjObf *is a secure black box obfuscator for conjunctions in the generic GES model.*

In fact, we prove a somewhat stronger statement. Whereas the definition of black box obfuscation requires to simulate adversaries with single-bit output, our simulator will be able to successfully simulate any polynomial time generic adversary, regardless of the number of bits it outputs.

Proof. Fix a security parameter λ, and the obfuscation of a conjunction C. The obfuscation is generated by choosing parameters for a graded encoding scheme, and then picking ring elements for all encodings. For each $i \in [n], b \in \{0,1\}$, the obfuscator

picks $\rho_{i,b}, \alpha_{i,b} \in R$, and also picks $\rho_{n+1}, \alpha_{n+1} \in R$. *Note that these ring elements are not all independently distributed* (though each of them, on its own, is uniformly random in R). For the security proof, we treat the encoded ring elements as random variables. We use $\vec{\rho}$ to denote the vector containing $\{\rho_{i,b}\}_{i\in[n+1],b\in\{0,1\}}$, and we call these the ρ random variables. We use $\vec{\alpha}$ to denote the vector containing $\{\alpha_{i,b}\}_{i\in[n+1],b\in\{0,1\}}$, and we call these the α random variables. (For notational convenience, we use the notation $\rho_{n+1,0} = \rho_{n+1,1} = \rho_{n+1}$, and $\alpha_{n+1,0} = \alpha_{n+1,1} = \alpha_{n+1}$).

For a conjunction $C = (W, V)$, consider the joint distribution of the $(\vec{\rho}, \vec{\alpha})$ random variables. The elements in $\vec{\rho}$ are all independent and uniformly random ring elements. For each $i \in [n]$, the $\alpha_{i,0}, \alpha_{i,1}$ RVs are either equal (if $i \in W$) or independent (if $i \notin W$). Finally, the α_{n+1} RV is always determined by the other α random variables ($\alpha_{n+1} = \prod_i \alpha_{i,V[i]}$).

The Distributions $D_{obf}(C)$ *and* D_{dummy}. Let $D_{obf}(C)$ denote the obfuscator's output distribution on the conjunction C:

$$\left(params, \{(w_{i,0}, u_{i,0}), (w_{i,1}, u_{i,1})\}_{i\in[n]}, (w_{n+1}, u_{n+1})\right)$$

where $(w_{i,b}, u_{i,b})$ are the encodings of $\rho_{i,b}$ and $(\rho_{i,b} \cdot \alpha_{i,b})$ (respectively) in $S_{\mathbf{e}_i}$, and (w_{n+1}, u_{n+1}) are the encodings of ρ_{n+1} and $(\rho_{n+1} \cdot \alpha_{n+1})$ (respectively) in $S_{\mathbf{e}_{n+1}}$. The obfuscator's output is determined by the choice of ring elements (with joint distribution as above), together with the coins (if any) used to randomize the encodings.[8]

In the security proof, we also consider a "dummy obfuscation" distribution D_{dummy}. This dummy distribution is generated by choosing all of the encoded ring elements independently and uniformly at random, and then outputting their encodings.

The Simulator $\mathcal{S}$. For a polynomial size generic adversary $\mathcal{A}$ (w.l.o.g. we assume that $\mathcal{A}$ is deterministic, otherwise sample a random tape for $\mathcal{A}$ and fix it throughout the simulation), the simulator $\mathcal{S}$ executes $\mathcal{A}$ on the dummy distribution D_{dummy} with a random graded encoding scheme (GES) oracle $\mathcal{RG}$ (where $\mathcal{RG}$ is simulated by $\mathcal{S}$). In particular, $\mathcal{S} = \mathcal{S}^{\mathcal{A}}$ can be viewed as a uniform machine with oracle access to $\mathcal{A}$.

For a conjunction $C = (W, V)$, consider the "real" distribution $\mathcal{A}(D_{obf}(C))$, where the randomness is over the choice of random GES oracle and the obfuscator's random coins, and a "simulated" distribution $\mathcal{A}(D_{\text{dummy}})$, where again the randomness is over the choice of random GES oracle and the choice of random ring elements for D_{dummy}. See Sect. 5.1 for the specification of a random GES oracle.

First, observe that in both cases, the obfuscation is identically distributed: the strings $\{(w_{i,b}, u_{i,b})\}$ and (w_{n+1}, u_{n+1}), which w.h.p. are all encodings of distinct indexed ring elements, are all independent and uniformly random (over the randomness of $\mathcal{RG}$). We emphasize that the indexed ring elements may not be independent, but their encodings under $\mathcal{RG}$ are nonetheless independent uniformly random strings. The difference between the two distributions is in $\mathcal{RG}$'s answers to the adversary's queries. For example, if the adversary computes the obfuscation's output on an input $\vec{x}$ for which $C(\vec{x}) = 1$,

[8] Recall that, as discussed in Remark 5.1, for security proofs in the random graded encoding scheme model, we assume that the encoding of each indexed ring element is unique.

then in $D_{obf}(C)$ this unveils a non-trivial multilinear equality between the indexed ring elements encoded in the obfuscation, whereas in D_{dummy} the indexed ring elements are independent, and there is no such multilinear equality between them. Thus, the oracle's answers allow the adversary to distinguish the two cases.

While this specific attack clearly demonstrates that the two distributions can be distinguished, it requires "knowing" an input $\vec{x}$ for which $C(\vec{x}) = 1$. We show that *this is (essentially) always the case*: for each oracle call made by $\mathcal{A}$, either its output is identically distributed in $D_{obf}(C)$ and in D_{dummy} (even conditioned on past answers and table state) or the call (implicitly) specifies a polynomial size set X of inputs, such that unless $\exists \vec{x} \in X, C(\vec{x}) = 1$, the call's output is uniformly random and identically distributed in $D_{obf}(C)$ and in D_{dummy}.

The high-level strategy of $\mathcal{S}$ is as follows. It feeds $\mathcal{A}$ with an obfuscation from D_{dummy}. As noted above, the distributions of the obfuscation and the initial state of the table T in $D_{obf}(C)$ and D_{dummy} are identical. The simulator now monitors each call made by $\mathcal{A}$ to $\mathcal{RG}$. For each such call, either its output and the updated table state are identically distributed in $D_{obf}(C)$ and in D_{dummy} or the call specifies a set of inputs X s.t. for some $\vec{x} \in X, C(\vec{x}) = 1$. Moreover, for each call, $\mathcal{S}$ can efficiently find such a set X, and then use its black box access to C to test, for every $\vec{x} \in X$, whether $\vec{x}$ is an accepting input and $C(\vec{x}) = 1$. If $\mathcal{S}$ finds an accepting input, it is essentially done—it can use its black box access to C to retrieve W and V (see Claim 3.3), and continue the simulation *perfectly* as in $D_{obf}(C)$. If there is no $\vec{x} \in X$ for which $C(\vec{x}) = 1$, then the output and the updated table state are identically distributed in $D_{obf}(C)$ and in D_{dummy}, and the simulator can continue its simulation as in D_{dummy}. Thus, the simulator can process $\mathcal{A}$'s calls one by one, and generate a view that is statistically close to $\mathcal{A}(D_{obf}(C))$.

We now elaborate on the simulator's operation, and its procedures for identifying the implicit inputs specified by $\mathcal{A}$'s oracle calls.

Characterizing $\mathcal{A}$'s Oracle Calls. We examine $\mathcal{A}$'s oracle calls in sequence, and consider $\mathcal{A}$'s view when it is run on $D_{obf}(C)$ and on D_{dummy}. For the add, negate, mult procedures, the indexed ring element in their output is a multilinear function of the $\vec{\rho}$ and $\vec{\alpha}$ random variables. For the isZero procedure, the output is a multilinear equality test on the random variables. In fact, these are multilinear functions of $(\vec{\rho}, \vec{\alpha})$ with a very specific structure. We call these *cross-linear* functions, see Definition 5.8 and Claim 5.9 below. The difference between $D_{obf}(C)$ and D_{dummy} is that these cross-linear functions may take different values because the variables in $\vec{\alpha}$ have a different distribution in $D_{obf}(C)$ and D_{dummy}. As shown in the example above, running the obfuscated program on an input $\vec{x}$ such that $C(\vec{x}) = 1$ ended in an isZero call specifying a multilinear constraint that was true when the random variables are distributed as in $D_{obf}(C)$, but false in D_{dummy}.

We proceed as follows. Recall that the oracle $\mathcal{RG}$ maintains a table T of all indexed ring elements whose encodings have already been specified (see Sect. 5.1). For each element in the table T, we consider its representation as a (cross-linear) function of the $(\vec{\rho}, \vec{\alpha})$ random variables. After the obfuscation (real or dummy) is specified, the entries in T contain the indexed elements $\rho_{i,b}$ and $\rho_{i,b} \cdot \alpha_{i,b}$. As $\mathcal{A}$ makes additional oracle calls, new entries are added to the table T. For these new entries, we consider the representations of their indexed ring elements as cross-linear functions of $(\vec{\rho}, \vec{\alpha})$. We

focus here on calls to the add, negate, mult, encRand procedures (calls to samp and isZero are handled similarly). For each such call, the inputs should be encodings in the table T: for any other string, the probability that it is a valid encoding is negligible, and the oracle $\mathcal{RG}$ just answers $\perp$ (in both the real and simulated executions). Otherwise, the inputs are in the table T, and the output may form a new entry in T. The *functional representation* of a table entry is defined as follows: □

Definition 5.4. (*Functional Representation*) For each entry in $\mathcal{RG}$'s table T, with indexed ring element $(\mathbf{v}, \sigma) \in \{0,1\}^\tau \times R$ and encoding $s \in \{0,1\}^*$, its *functional representation* $f(\vec{\rho}, \vec{\alpha})$ is defined recursively:

1. Initially, the only table entries are encodings that appear in the obfuscation, i.e., the entries for the ring elements $\{(\rho_{i,b}, (\rho_{i,b} \cdot \alpha_{i,b}))\}_{i\in[n],b\in\{0,1\}}$, where the variables associated with i are indexed by $\mathbf{e}_i$. These table entries are all distinct (excet with negligible probability), and for each table entry its functional representation is simply $f(\vec{\rho}, \vec{\alpha}) = \rho_{i,b}$ or $f(\vec{\rho}, \vec{\alpha}) = (\rho_{i,b} \cdot \alpha_{i,b})$ (respectively).
2. For subsequent entries that are created on $\mathcal{A}$'s calls to $\mathcal{RG}$, their functional representation is defined recursively, e.g. for an add call, with input encodings s_1 and s_2, if s_1 or s_2 is not in $\mathcal{RG}$'s table T, then that input does not represent a valid encoding, the output is $\perp$ and no new table entry is created. If the inputs are in the table T, let $(\mathbf{v}_1, \sigma_1)$ and $(\mathbf{v}_2, \sigma_2)$ be the indexed ring elements that they encode. If $\mathbf{v}_1 \neq \mathbf{v}_2$ then again the output is $\perp$ and no new table entry is created. The remaining case is $\mathbf{v}_1 = \mathbf{v}_2 = \mathbf{v}$. In this case, let f_1 and f_2 be the functional representations of the input encodings. If a new table entry is created for the output, then its representation is $(f_1 + f_2)$.
   The cases of samp, encRand, negate, mult calls are handled similarly.

Remark 5.5. We emphasize that, for a table entry t, its functional representation is completely independent of the conjunction being obfuscated. Indeed, the functional representation remains unchanged even if the $(\vec{\rho}, \vec{\alpha})$ random variables are drawn uniformly at random, as in D_{dummy}.

By definition, the functional representation indeed computes the ring element in a table entry (for any setting of $(\vec{\rho}, \vec{\alpha})$). Moreover, for any table entry, $\mathcal{S}$ can compute a polynomial size arithmetic circuit computing that table entry's functional representation:

Claim 5.6. *For any setting of the initial random variables $(\vec{\rho}, \vec{\alpha})$, and for an entry t in the table T containing the indexed ring element $(\mathbf{v}_t, \sigma_t)$ and with functional representation f_t, it is the case that $\sigma_t = f_t(\vec{\rho}, \vec{\alpha})$.*

Claim 5.7. *For each entry t in $\mathcal{RG}$'s table T, the simulator $\mathcal{S}$ can compute a polynomial size arithmetic circuit computing its functional representation $f_t(\vec{\rho}, \vec{\alpha})$.*

Proof Sketch. The proof is by induction, following the recursive structure of Definition 5.4:

1. *Induction basis* for the obfuscation encodings that are initially in table T, $\mathcal{S}$ can compute the functional representation ($\rho_{i,b}$ or $\rho_{i,b} \cdot \alpha_{i,b}$) for each of these table entries, and a polynomial size arithmetic circuit that computes it.
2. *Induction step* for one of $\mathcal{A}$'s calls to $\mathcal{RG}$, say an add call, as in Definition 5.4, if $\mathcal{S}$ knows functional representations f_1 and f_2 of the input arguments (and poly-sized arithmetic circuits that compute them), then it can also compute the output's functional representation ($f_1 + f_2$) (and a poly-sized arithmetic circuit that computes it). □

As noted above, the functional representations of all table entries are multilinear functions on $(\vec{\rho}, \vec{\alpha})$. Moreover, they are multilinear functions with a specific structure, which we call *cross-linear* functions:

Definition 5.8. (*cross-linear function*) A function g is a *cross-linear term over* $(\vec{\rho}, \vec{\alpha})$ if it is of the form:

$$g = \gamma \cdot \prod_{i \in I} \rho_{i,\vec{x}[i]} \cdot \prod_{i \in I \cap A} \alpha_{i,\vec{x}[i]}$$

where $I, A \subseteq [n+1]$, $\vec{x} \in \{0,1\}^{n+1}$, and $\gamma \in R$.

A function f is a *cross-linear function of* $(\vec{\rho}, \vec{\alpha})$ if it can be expressed as a sum of cross-linear terms. I.e., it is of the form:

$$f = \sum_j \gamma_j \cdot \prod_{i \in I_j} \rho_{i,\vec{x}_j[i]} \cdot \prod_{i \in I_j \cap A_j} \alpha_{i,\vec{x}_j[i]}$$

where for each j: $I_j, A_j \subseteq [n+1]$, $\vec{x}_j \in \{0,1\}^{n+1}$, and $\gamma_j \in R$. We assume that the expansion is compact in the sense that the tuples $\{(I_j, A_j, \vec{x}_j)\}_j$ are all distinct. We call this particular expansion the *cross-linear expansion of* f.

Claim 5.9. *For any entry in the table T, its functional representation is a cross-linear function of* $(\vec{\rho}, \vec{\alpha})$.

Proof. We show that, for every entry in the table T, indexed by a vector $\mathbf{v}$, its functional representation is a cross-linear function of the ρ and α variables that are indexed by vectors $\mathbf{e}_i$ for which $\mathbf{v}[i] = 1$. The proof is by induction over the number of calls to the oracle $\mathcal{RG}$.

- For the induction basis, after the obfuscator completes its run, the only entries in the table T are for the ring elements $\{(\rho_{i,b}, (\rho_{i,b} \cdot \alpha_{i,b}))\}_{i \in [n], b \in \{0,1\}}$, where the variables associated with i are indexed by $\mathbf{e}_i$. For each such indexed ring element, its functional representation is of the form $\rho_{i,b}$ or $\rho_{i,b} \cdot \alpha_{i,b}$, which are both cross-linear functions over the variables indexed by $\mathbf{e}_i$.
- For the induction step, we first handle the case of add calls. The proofs for negate and encRand are similar. The proof for mult is slightly different, and follows below. For an add call, let s_1 and s_2 be the input encodings. As in Definition 5.4,

add's output is $\perp$ unless the encodings are both in the table T, and contain indexed ring elements $(\mathbf{v}, \sigma_1)$ and $(\mathbf{v}, \sigma_2)$. In this case, by induction, the functional representation of both input items is a cross-linear function of the α and ρ random variables indexed by vectors $\mathbf{e}_i$ for which $\mathbf{v}[i] = 1$. Thus, the functional representation of the sum is also a cross-linear function of these same variables.
For a call to mult, similarly to the above, the output is $\perp$ unless the inputs are encodings of table entries $(\mathbf{v}_1, \sigma_1)$ and $(\mathbf{v}_2, \sigma_2)$, and the supports of $\mathbf{v}_1$ and $\mathbf{v}_2$ (the entries on which they are 1) are disjoint. In this case, by the induction hypothesis, the functional representations of the input arguments are cross-linear functions of disjoint sets of ρ and α variables, and we conclude that the output's functional representation is a cross-linear function of the variables indexed by $\mathbf{e}_i$ for which $\mathbf{v}_1[i] = 1$ or $\mathbf{v}_2[i] = 1$. □

We note that while it is true that $\mathcal{S}$ can compute a polynomial size circuit computing each table entry's functional representation (Claim 5.7), it does not seem that the simulator can obtain the *cross-linear expansion* of these functions. Indeed, the cross-linear expansion may not have a polynomial size description.

Classifying $\mathcal{A}$'s Oracle Queries. From Claim 5.9 we know that the functional representations of all entries in T (and in particular the representations of all inputs and outputs to oracle calls) are cross-linear functions of $(\vec{\rho}, \vec{\alpha})$. For any fixed call and table state, the outputs' *functional representations* in the execution of $\mathcal{A}$ on $D_{obf}(C)$ and D_{dummy} are identical (see Remark 5.5). This does not, however, mean that *the outputs* are identically distributed: as noted above, it might be the case that when the random variables are sampled as in $D_{obf}(C)$, the dependencies between the random variables causes the functional representation to become equivalent to that of an item that's already in the table T; e.g., following the example outlined above, there are two distinct ways products of the random variables that evaluate to $\Pi\rho_{i,V[i]} \cdot \alpha_{i,V[i]}$. In this case, when the second product is evaluated, the output will be the encoding of an item that's already in the table. For that same call in the simulation on D_{dummy}, however, the random variables are independent, and the functional representation of the new output is different from all entries in the table. In this case, the output is a fresh uniformly random encoding.

This is the main challenge for the simulator $\mathcal{S}$. For each of $\mathcal{A}$'s calls to $\mathcal{RG}$, $\mathcal{S}$ "wants to know" whether the output is the same as an entry that's already in the table or not. We show that for a fixed table T, and an oracle call made by $\mathcal{A}$, there are only three possibilities:

Possibility 1: *Output (Always) Identical to Another Entry* This is the simplest case. Here, in both the real and simulated executions, the output's functional representation is equivalent to some other entry in T. Namely taking f_o to be the output's functional representation, there exists some entry t in T with functional representation f_t, such that $f_o \equiv f_t$. In this case, for both executions, the oracle outputs item t's encoding and the table remains unchanged.

We note that $(f_o - f_t)$ is a cross-linear function of $(\vec{\rho}, \vec{\alpha})$, and that for any cross-linear function there exists a simple procedure for testing whether $(f_o - f_t) \equiv 0$.

Claim 5.10. *Let f be a cross-linear function of $(\vec{\rho}, \vec{\alpha})$. There exists an efficient algorithm with black box access to f, such that if $f \not\equiv 0$, then with all but $negl(\lambda)$ probability, the algorithm outputs 1. Otherwise the algorithm outputs 0.*

Proof. The algorithm picks a uniformly random assignment for $(\vec{\rho}, \vec{\alpha})$ in $R = \mathbb{Z}_p$, and checks whether f is 0 on this assignment. If so, it outputs 0, otherwise it outputs 1. Since f is cross-linear, its total degree is at most $2(n+1)$. By the Schwartz-Zippel Lemma, if $f \not\equiv 0$ then the algorithm will output 0 with probability at most $O(n^\sigma/|R|) = negl(\lambda)$. □

To recognize calls of type I, the simulator operates as follows: for each of $\mathcal{A}$'s oracle calls, it computes the output's functional representation and then tests whether is equivalent to the functional representation of any table entry. If so, $\mathcal{S}$ passes the call to the oracle, returns the output to $\mathcal{A}$, and continues the simulation without changing T.

Possibility II: *Output (Always) Different from All Entries* Here, in both the real and simulated executions, the output's functional representation is not equivalent to that of any other entry in T. In this case, for both distributions, the oracle outputs a fresh, uniformly random encoding for that output and updates the table to include the new item.

Below, we show a simple condition that the simulator can use efficiently identify calls of this type. Once such a call is recognized, the simulator can simply pass the call to the oracle, return the output to $\mathcal{A}$, and update its view of the table T to include the new indexed item and its functional representation.

Possibility III: *Output Varies* This is the trickiest case: in the real execution, the output's functional representation is equivalent to some other entry in T, but in the simulated execution, it is not equivalent to that entry. This is problematic case, because $\mathcal{S}$ doesn't know what the output distribution is in the real execution, and can't continue the simulation.

We show that whenever this case occurs, the adversary's call specifies an efficiently recoverable input $\vec{x} \in \{0,1\}^n$ such that $C(\vec{x}) = 1$. $\mathcal{S}$ can therefore find this $\vec{x}$ and then use its C-oracle to recover C and complete the simulation.

Answering $\mathcal{A}$'s Oracle Queries. For a successful simulation, $\mathcal{S}$ needs to distinguish and handle cases II and III above. For this, we use the following definitions:

Definition 5.11. (*ρ-Monomials, full monomials*) A function f is a ρ-monomial if it is a cross-linear term in $\vec{\rho}$ (i.e., independent of $\vec{\alpha}$). We say that a ρ-monomial is *full* if its total degree is exactly $n+1$ (i.e., it has maximal total degree). In this case $f = ((\prod_{i\in[n]} \rho_{i,\vec{x}[i]}) \cdot \rho_{n+1,0})$, for some $\vec{x} \in \{0,1\}^n$, and we say that f is the full ρ-monomial $\vec{x}$.

Definition 5.12. (*Cross-linear function with a full ρ-monomial*) We say that a cross-linear function f of $(\vec{\rho}, \vec{\alpha})$ *includes a full ρ-monomial*, if in f's cross-linear expansion, there exists a j such that $\gamma_j \neq 0$ and I_j is of size exactly $(n+1)$ (and so the j-th term in the expansion is a full ρ-monomial).

We show that for any cross-linear function f, if it includes a non-full ρ-monomial, then (for any conjunction C) when $(\vec{\rho}, \vec{\alpha})$ are drawn as in $D_{obf}(C)$, with all but negligible probability $f(\vec{\rho}, \vec{\alpha}) \neq 0$. Moreover, if f *does* include a full ρ-monomial $\vec{x}$, then for any conjunction C, if $C(\vec{x}) = 0$, then when $(\vec{\rho}, \vec{\alpha})$ are drawn as in $D_{obf}(C)$, with all but negligible probability $f(\vec{\rho}, \vec{\alpha}) \neq 0$. This is shown in Claims 5.13 and 5.14. Finally, we show that there exists an efficient procedure for identifying a ρ-monomial of a function f. This is in Claim 5.15.

$\mathcal{S}$'s strategy for simulating $\mathcal{A}$'s oracle calls that are not of type I follows from these claims. For each such call, whose output has functional representation f_o, for each entry t in the table T with functional representation f_t, $\mathcal{S}$ can identify a ρ-monomial of $(f_o - f_t)$: if it finds a non-full monomial, then by Claim 5.13, w.h.p. in both $D_{obf}(C)$ and D_{dummy}, the indexed ring element in the call's output is not equal to the element in entry t of the table. Otherwise, if $\mathcal{S}$ finds a full ρ monomial $\vec{x}_t$ of $(f_o - f_t)$, then it uses its black box access to C to test whether $C(\vec{x}_t) = 1$. If so, $\mathcal{S}$ can further use its black box access to extract the representation (W, V) of C and complete the simulation perfectly. If not, then by Claim 5.14, w.h.p. in both $D_{obf}(C)$ and D_{dummy}, the indexed ring element in the call's output is not equal to the element in entry t. Finally, if after testing all entries in the table, $\mathcal{S}$ has not found an accepting input, then the call is of type II and the simulation can continue.

The claims follow:

Claim 5.13. *Let f be a cross-linear function of $(\vec{\rho}, \vec{\alpha})$ s.t. $f \not\equiv 0$. If f includes a non-full ρ-monomial, then for any conjunction C, for an assignment to $(\vec{\rho}, \vec{\alpha})$ drawn as in $D_{obf}(C)$, it holds that* $\Pr_{(\vec{\rho},\vec{\alpha})}[f(\vec{\rho}, \vec{\alpha}) = 0] = negl(\lambda)$.

Proof. Let $g(\vec{\rho}) = \Pi_{i \in S} \rho_{i,\vec{x}[i]}$ be a non-full monomial, with $I \subsetneq [n+1]$ and $\vec{x} \in \{0,1\}^{n+1}$. Then, by definition, the cross-linear expansion of f can be written as a sum:

$$f(\vec{\alpha}, \vec{\rho}) = (\gamma \cdot g(\vec{\rho}) \cdot h(\vec{\alpha})) + \sum_{j:((I_j \neq I) \vee (\vec{x}_j \neq \vec{x}))} \gamma_j \cdot \prod_{i \in I_j} \rho_{i,\vec{x}_j[i]} \cdot h_j(\vec{\alpha})$$

where $\gamma \neq 0$, and where $h(\vec{\alpha}) \not\equiv 0$ can be expanded as:

$$h(\vec{\alpha}) = \sum_k \delta_k \cdot \prod_{i \in I \cap A_k} \alpha_{i,\vec{x}[i]}$$

and each A_k is a (distinct) subset of I.

Now consider $h(\vec{\alpha})$, where the $\vec{\alpha}$ random variables are chosen as in $D_{obf}(C)$, for $C = (W, V)$. This means that the $\vec{\alpha}$ random variables are not independent, but rather for some i's it may be that $\alpha_{i,0} = \alpha_{i,1}$, and $\alpha_{n+1} = \prod_i \alpha_{i,V[i]}$. Other than these restrictions the random variables in $\vec{\alpha}$ are independent and uniformly random. In particular, this means that the random variables on which h depends, namely $\{\alpha_{i,\vec{x}[i]}\}_{i \in I \subsetneq [n+1]}$, are all independent and uniformly random (or rather within statistical distance $negl(\lambda)$ of independent and uniformly random).[9] Thus h is a nonzero multilinear function of

[9] The only dependence is that if for $i \in [n]$ any of the $\vec{\alpha}_{i,\vec{x}[i]}$ variables are 0, then α_{n+1} is also 0.

independent uniformly random variables, and by the Schwartz-Zippel Lemma with all but $negl(\lambda)$ probability over the choice of $\vec{\alpha}$ by $D_{obf}(C)$, we have that $h(\vec{\alpha}) \neq 0$.

Drawing $\vec{\alpha}$ as in $D_{obf}(C)$, and setting those variables in f, we examine the expansion of f. We get that:

$$f_{\vec{\alpha}}(\vec{\rho}) = (\gamma' \cdot g(\vec{\rho})) + \sum_{j:((I_j \neq I) \vee (\vec{x}_j \neq \vec{x}))} \delta'_j \cdot \prod_{i \in I_j} \rho_{i,\vec{x}_j[i]}$$

where with all but $negl(\lambda)$ probability over $\vec{\alpha}$, we have that $\gamma' \neq 0$. In this case, since the $\vec{\rho}$ random variables are all independent and uniformly random, the $g(\vec{\rho})$ monomial in the expansion cannot be canceled out and $f_{\vec{\alpha}}(\vec{\rho}) \not\equiv 0$. Thus when we pick $\vec{\rho}$ uniformly at random (as in $D_{obf}(C)$), with all but $negl(\lambda)$ probability we have $0 \neq f_{\vec{\alpha}}(\vec{\rho}) = f(\vec{\rho}, \vec{\alpha})$. □

Claim 5.14. *Let f be a cross-linear function of $(\vec{\rho}, \vec{\alpha})$ s.t. $f \not\equiv 0$. Suppose f includes a full ρ-monomial $\vec{x} \in \{0,1\}^n$. For any conjunction C, if $C(\vec{x}) = 0$, then for an assignment on $(\vec{\rho}, \vec{\alpha})$ drawn as in $D_{obf}(C)$, $\Pr_{(\vec{\rho},\vec{\alpha})}[f(\vec{\rho}, \vec{\alpha}) = 0] = negl(\lambda)$.*

Proof. The proof is similar to that of Claim 5.13. Let $g(\vec{\rho}) = \Pi_{i \in [n+1]} \rho_{i,\vec{x}[i]}$ be the full ρ-monomial (we take $\vec{x}[n+1] = 0$ for notational convenience). Then, by definition, the cross-linear expansion of f can be written as a sum:

$$f(\vec{\alpha}, \vec{\rho}) = (\gamma \cdot g(\vec{\rho}) \cdot h(\vec{\alpha})) + \sum_{j:((I_j \neq [n+1]) \vee (\vec{x}_j \neq \vec{x}))} \gamma_j \cdot \prod_{i \in I_j} \rho_{i,\vec{x}_j[i]} \cdot h_j(\vec{\alpha})$$

where $\gamma \neq 0$ and $h(\vec{\alpha}) \not\equiv 0$ can be expanded as:

$$h(\vec{\alpha}) = \sum_k \gamma_k \cdot \prod_{i \in A_k} \alpha_{i,\vec{x}[i]}$$

where each A_k is a (distinct) subset of $[n+1]$.

As in Claim 5.13, if $C(\vec{x}) = 0$, then the input variables to h are (close to) independent and uniformly random even under $D_{obf}(C)$. Thus w.h.p. over the choice of $\vec{\alpha}$ as in $D_{obf}(C)$, we have $h(\vec{\alpha}) \neq 0$. Fixing any such α, we get that $f_{\vec{\alpha}}(\vec{\rho}) \not\equiv 0$, and so w.h.p. over the choice of $\vec{\rho}$ we get: $0 \neq f_{\vec{\alpha}}(\vec{\rho}) = f(\vec{\rho}, \vec{\alpha})$. □

Claim 5.15. *Let f be a cross-linear function on $(\vec{\rho}, \vec{\alpha})$. There is an efficient algorithm with black box access to f, such that if f is not a constant function, then with all but $negl(\lambda)$ probability, the algorithm outputs a ρ-monomial of f.*

Proof. The proof proceeds as follows. Initially set $f_0 = f$, $I = \Phi$, and $i = 0$. Proceed in iterations $i \leftarrow 0, \ldots, n$:
In the i-th iteration, set both $\rho_{i,0}$ and $\rho_{i,1}$ to 0 in f_i, and check whether the resulting function is identically 0 (using the algorithm of Claim 5.10):

- If not, then there must exist a ρ-monomial in f_i that includes neither $\rho_{i,0}$ nor $\rho_{i,1}$. The algorithm sets $\vec{x}[i] = 0$, sets f_{i+1} to be equal to f_i, but with the variables $\rho_{i,0}$, $\rho_{i,1}$ set to 0, and proceeds to iteration $(i + 1)$.
- Otherwise (the resulting function above was not identically equal to 0), then there must exist a ρ-monomial in f_i that includes either $\rho_{i,0}$ or $\rho_{i,1}$. Similarly to the above, the algorithm can determine whether there exists ρ-monomial that includes $\rho_{i,0}$ (by setting $\rho_{i,1}$ to 0 and checking that the resulting function is not always 0). If such a monomial exists, then the algorithm adds i to the set I, sets $\vec{x}[i] = 0$, sets f_{i+1} to be equal to f_i, but setting $\rho_{i,0} = 1$ and $\rho_{i,1} = 0$, and proceeds to iteration $(i + 1)$.
- Finally, in the remaining case, there exists a monomial that includes $\rho_{i,1}$, the algorithm adds i to the set I, sets $\vec{x}[i] = 1$, sets f_{i+1} to be equal to f_i, but setting $\rho_{i,0} = 0$ and $\rho_{i,1} = 1$, and proceeds to the next iteration.

The algorithm's output is the ρ-monomial $\prod_{i \in I} \rho_{i,\vec{x}[i]}$, which (by construction) is a ρ-monomial of f. Observe that unless f is a constant function, with all but negligible probability the algorithm identifies a ρ-monomial. □

Acknowledgements

We thank the reviewers of CRYPTO 2013 for their comments. We thank Ryo Nishimaki for pointing us to the zeroing attack on [21], and the authors of [14] for discussing the applicability of such attack on their scheme.

References

[1] B. Applebaum, Z. Brakerski, Obfuscating circuits via composite-order graded encoding, in Y. Dodis, J. B. Nielsen, editors, *Theory of Cryptography—12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23–25, 2015, Proceedings, Part II*. Lecture Notes in Computer Science, vol. 9015 (Springer, 2015), pp. 528–556

[2] B. Adida, D. Wikström, How to shuffle in public, in *Vadhan [34]*, pp. 555–574

[3] B. Barak, N. Bitansky, R. Canetti, Y. T. Kalai, O. Paneth, A. Sahai, Obfuscation for evasive functions, in Y. Lindell, editor, *Theory of Cryptography—11th Theory of Cryptography Conference, TCC 2014, San Diego, CA, USA, February 24–26, 2014. Proceedings*. Lecture Notes in Computer Science, vol. 8349 (Springer, 2014), pp. 26–51

[4] N. Bitansky, R. Canetti, On strong simulation and composable point obfuscation, in *CRYPTO* (2010), pp. 520–537

[5] A. Boldyreva, S. Fehr, A. O'Neill, On notions of security for deterministic encryption, and efficient constructions without random oracles, in D. Wagner, editor, *CRYPTO*. Lecture Notes in Computer Science, vol. 5157 (Springer, 2008), pp. 335–359

[6] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. P. Vadhan, K. Yang, On the (im)possibility of obfuscating programs. *J. ACM*, **59**(2), 6 (2012). Preliminary version in CRYPTO 2001

[7] Z. Brakerski, G. N. Rothblum, Black-box obfuscation for d-cnfs, in *Innovations in Theoretical Computer Science, ITCS'14, Princeton, NJ, USA, January 12–14, 2014* (2014), pp. 235–250

[8] Z. Brakerski, G. N. Rothblum, Virtual black-box obfuscation for all circuits via generic graded encoding, in *Theory of Cryptography—11th Theory of Cryptography Conference, TCC 2014, San Diego, CA, USA, February 24–26, 2014. Proceedings* (2014), pp. 1–25

[9] D. Boneh, A. Silverberg, Applications of multilinear forms to cryptography. *IACR Crypt. ePrint Arch.***2002**, 80 (2002)
[10] R. Canetti, Towards realizing random oracles: Hash functions that hide all partial information. in *CRYPTO* (1997), pp. 455–469
[11] R. Canetti, R. R. Dakdouk, Obfuscating point functions with multibit output, in *EUROCRYPT* (2008), pp. 489–508
[12] J.-S. Coron, C. Gentry, S. Halevi, T. Lepoint, H. K. Maji, E. Miles, M. Raykova, A. Sahai, M. Tibouchi, Zeroizing without low-level zeroes: new MMAP attacks and their limitations. *IACR Crypt. ePrint Arch.***2015**, 596 (2015)
[13] J. H. Cheon, K. Han, C. Lee, H. Ryu, D. Stehlé, Cryptanalysis of the multilinear map over the integers, in E. Oswald, M. Fischlin, editors, *Advances in Cryptology—EUROCRYPT 2015—34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26–30, 2015, Proceedings, Part I*. Lecture Notes in Computer Science, vol. 9056 (Springer, 2015), pp. 3–12
[14] J.-S. Coron, T. Lepoint, M. Tibouchi, Practical multilinear maps over the integers, in R. Canetti, J. A. Garay, editors, *Advances in Cryptology—CRYPTO 2013—33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2013. Proceedings, Part I*. Lecture Notes in Computer Science, vol. 8042 (Springer, 2013), pp. 476–493
[15] J.-S. Coron, T. Lepoint, M. Tibouchi, New multilinear maps over the integers. *IACR Crypt. ePrint Arch.***2015**, 162 (2015)
[16] R. Canetti, D. Micciancio, O. Reingold, Perfectly one-way probabilistic hash functions (preliminary version), in *STOC* (1998), pp. 131–140
[17] R. Canetti, G. N. Rothblum, M. Varia. Obfuscation of hyperplane membership, in *TCC* (2010), pp. 72–89
[18] Y. Dodis, R. Ostrovsky, L. Reyzin, A. Smith, Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM J. Comput.***38**(1), 97–139 (2008). Preliminary version in Eurocrypt 2004
[19] Y. Dodis, A. Smith, Correcting errors without leaking partial information, in *Gabow and Fagin [20]*, pp. 654–663
[20] H. N. Gabow, R. Fagin, editors. *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22–24, 2005*. ACM (2005)
[21] S. Garg, C. Gentry, S. Halevi, Candidate multilinear maps from ideal lattices, in T. Johansson, P. Q. Nguyen, editors, *EUROCRYPT*. Lecture Notes in Computer Science, vol. 7881 (Springer, 2013) pp. 1–17. See also Cryptology ePrint Archive, Report 2012/610
[22] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, B. 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* IEEE Computer Society (2013) pp. 40–49
[23] S. Goldwasser, Y. T. Kalai, On the impossibility of obfuscation with auxiliary input, in *FOCS* (2005) pp. 553–562
[24] S. Goldwasser, G. N. Rothblum, On best-possible obfuscation, in *Vadhan [34]*, pp. 194–213
[25] Y. Hu, H. Jia, Cryptanalysis of ggh map. *Crypt. ePrint Arch.* Report 2015/301 (2015). http://eprint.iacr.org/
[26] D. Hofheinz, J. Malone-Lee, M. Stam, Obfuscation for cryptographic purposes. *J. Cryptol.***23**(1), 121–168 (2010)
[27] S. Hohenberger, G. N. Rothblum, A. Shelat, V. Vaikuntanathan. Securely obfuscating re-encryption. *J. Cryptol.***24**(4), 694–719 (2011)
[28] M. J. Kearns, U. V. Vazirani, *An Introduction to Computational Learning Theory* (MIT Press, Cambridge, MA, USA, 1994)
[29] B. Lynn, M. Prabhakaran, A. Sahai, Positive results and techniques for obfuscation, in *EUROCRYPT* (2004), pp. 20–39
[30] U. M. Maurer, Abstract models of computation in cryptography, in *IMA International Conference* (2005), pp. 1–12
[31] R. Rothblum, On the circular security of bit-encryption, in *TCC* (2013), pp. 579–598
[32] R. Renner, S. Wolf, Smooth renyi entropy and applications, in *IEEE International Symposium on Information Theory, IEEE International Symposium on Information Theory* (2004), p. 233

[33] V. Shoup, Lower bounds for discrete logarithms and related problems, in *EUROCRYPT* (1997), pp. 256–266
[34] S. P. Vadhan, editor. *Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21–24, 2007, Proceedings*. Lecture Notes in Computer Science, vol. 4392 (Springer, 2007)
[35] H. Wee, On obfuscating point functions, in *Gabow and Fagin [20]*, pp. 523–532