More on Impossibility of Virtual Black-Box Obfuscation in Idealized Models

Mohammad Mahmoody[*] Ameer Mohammed[†] Soheil Nematihaji[‡]

June 25, 2015

Abstract

The celebrated work of Barak et al. (Crypto'01) ruled out the possibility of *virtual black-box* (VBB) obfuscation for general circuits. The recent work of Canetti, Kalai, and Paneth (TCC'15) extended this impossibility to the random oracle model as well assuming the existence of trapdoor permutations (TDPs). On the other hand, the works of Barak et al. (Crypto'14) and Brakerski-Rothblum (TCC'14) showed that general VBB obfuscation is indeed possible in *idealized graded encoding* models. The recent work of Pass and Shelat (Cryptology ePrint 2015/383) complemented this result by ruling out general VBB obfuscation in idealized graded encoding models that enable evaluation of constant-degree polynomials in finite fields.

In this work extend the above two impossibly results for general VBB obfuscation in idealized models. In particular we prove the following two results both assuming the existence of TDPs:

- There is no general VBB obfuscation in the generic group model of Shoup (Eurocrypt'97) for *any abelian* group. By applying our techniques to the setting of Pass and Shelat we extend their result to any (even non-commutative) finite *ring*.
- There is no general VBB obfuscation even in the *random trapdoor permutation oracle* model. Our proof extends to any number of levels of hierarchical trapdoors.

Keywords: Virtual Black-Box Obfuscation, Idealized Models.

[*]University of Virginia, `mohammad@cs.virginia.edu`. Supported by NSF CAREER award CCF-1350939.
[†]University of Virginia, `am8zv@virginia.edu`. Supported by University of Kuwait.
[‡]University of Virginia, `sn8fb@virginia.edu`. Supported by NSF award CCF-1350939.

Contents

1 Introduction

Obfuscating programs to make them "unintelligible" while preserving their functionality is one of the most sought after holy grails in cryptography due to its numerous applications. The celebrated work of Barak et al. [BGI+01] was the first to launch a formal study of this notion in its various forms. *Virtual Black-Box* (VBB) obfuscation is a strong form of obfuscation in which the obfuscated code does not reveal any secret bit about the obfuscated program unless that information could already be obtained through a black-box access to the program. It was shown in [BGI+01] that VBB obfuscation is not possible *in general* as there is a family of functions $\mathcal{F}$ that could not be VBB obfuscated. Roughly speaking, $\mathcal{F}$ would consist of circuits C such that given any obfuscation $B = Obf(C)$ of C, by running B over B itself as input one can obtain a secret s about C that could not be obtained through mere black-box interaction with C. This strong impossibility result, however, did not stop the researchers from exploring the possibility of VBB obfuscation for *special* classes of functions, and positive results for special cases were presented (e.g., [Can97, Wee05]) based on believable computational assumptions.

The work of Lynn, Prabhakaran and Sahai [LPS04] showed the possibility of VBB obfuscation for certain class of functions in the *random oracle model* (ROM). The work of [LPS04] left open whether general purpose obfuscator for all circuits could be obtained in the ROM or not. Note that when we allow the random oracle to be used during the obfuscation phase (and also let the obfuscated code to call the random oracle) the impossibility result of [BGI+01] no longer applies, because the proof of [BGI+01] requires the obfuscated algorithm to be a circuit in the *plain* model where no oracle is accessed. In fact, despite the impossibility of general VBB obfuscation in the plain model, a construction for VBB obfuscation in the ROM could be used as a practical heuristic obfuscation mechanism once instantiated with a strong hash function such as SHA3. This would be in analogy with the way ROM based constructions of other primitives are widely used in practice *despite* the impossibility results of [CGH04].

On a different route, the breakthrough of Garg et al. [GGH+13b] proposed a candidate *indistinguishability obfuscation* (iO), a weaker form of obfuscation compared to VBB for which no impossibility results were (are) known, relying on the so called "approximate multi-linear maps" (MLM) [GGH13a]. Shortly after, it was proved by Barak et al. [BGK+14] and Brakerski and Rothblum [BR14] that the construction of [GGH+13b] could be used to get even VBB secure obfuscation (rather than the weaker variant of iO) in an idealized form of MLMs, called the *graded encoding model*. The VBB obfuscation schemes of [BGK+14, BR14] proved in idealized models raised new motivations for studying the possibility of VBB obfuscation in such models of computation including the ROM.

Canetti, Kalai, and Paneth [CKP15] proved the first impossibility result for VBB obfuscation in a natural idealized model by ruling out the existence of general purpose VBB obfuscators in the random oracle model, assuming the existence of trapdoor permutations. Their work resolved the open question of [LPS04] negatively. At a technical level, [CKP15] showed how to compile any VBB obfuscator in the ROM into an *approximate* VBB obfuscator in the plain model which preserves the circuit's functionality only for "most" of the inputs. This would rule out VBB obfuscation in plain model (assuming TDPs) since previously, Bitansky and Paneth [BP13] had shown that no approximate VBB obfuscator for general circuits exist if trapdoor permutations exist.

Pass and shelat [Pas15] further studied the possibility of VBB obfuscation in idealized algebraic models in which the positive results of [BGK+14, BR14] were proved. [Pas15] showed that the existence of VBB obfuscation schemes in the graded encoding model, highly depends on the degree of polynomials (allowed to be zero tested) in this model. In particular they showed that VBB obfuscation of general circuits is impossible in the graded encoding model of constant-degree polynomials. Their work nicely complemented the positive results of [BGK$^+$14, BR14] that were proved in a similar (graded encoding) model but using super-constant (in fact polynomial in security parameter) polynomials.

We shall emphasize that proving *limitations* of VBB obfuscation or any other primitive in generic models of computation such as the generic group model of Shoup [Sho97] are strong lower-bounds (a la black-box separations [RTV04, IR89]) since such results show that for certain crypto tasks, as long as one uses certain algebraic structures (e.g., an abelian group) in a black-box way as the source of computational hardness, there will always be a *generic* attack that (also treats the underlying algebraic structure in a black-box way and) breaks the constructed scheme. The fact that the proposed attack is generic makes the lower-bound only stronger.

1.1 Our Results

In this work we extend the previous work [CKP15, Pas15] on the impossibility of VBB obfuscation in idealized models of computation and generalize their results to more powerful idealized primitives. We first focus on the generic group model of Shoup [Sho97] (see Definitions 2.5 and 2.6) and rule out the existence of general VBB obfuscation in this model for *any finite abelian group.*

Theorem 1.1. *Assuming the existence of trapdoor permutations, there exists no virtual black-box obfuscation for general circuits in the generic group model for any finite abelian group.*

The work of [Pas15] implies a similar lower bound for the case of abelian groups of prime order. We build upon the techniques of [CKP15, Pas15] and extend the result of [Pas15] to arbitrary (even noncyclic) finite abelian groups. See the next section for a detailed description of our techniques for proving this theorem and the next theorems described below.

We then apply our techniques designed to prove Theorem 1.1 to the setting of graded-encoding model studied in [Pas15] and extend their results to arbitrary finite *rings* (rather than fields) which remained open after their work. Our proof even handles *noncommutative* rings.

Theorem 1.2. *Assuming the existence of trapdoor permutations, there exists no virtual black-box obfuscation for general circuits in degree-$O(1)$ ideal graded encoding model for* any *finite ring R.*

Finally, we generalize the work of [CKP15] beyond random oracles by ruling out general VBB obfuscation in random *trapdoor permutations* (TDP) oracle model. Our result extends to an arbitrary number of levels of hierarchy of trapdoors, capturing primitives such as ideal hierarchical identity based encryption [HL02].

Theorem 1.3. *Assuming the existence of trapdoor permutations, there exists no virtual black-box obfuscation for general circuits in the idealized model of random trapdoor permutations, even if the oracle provides an unbounded* hierarchy *of trapdoors.*

Note that the difference between the power of random oracles and random TDPs in cryptography is usually huge, as random oracle is an idealized primitive giving rise to (very efficient) *symmetric* key cryptography primitives, while TDPs could be used to construct many *public-key* objects. Our result indicates that when it comes to VBB obfuscation random TDPs are no more powerful than just random oracles.

1.2 Technical Overview

The high level structure of the proofs of our results follows the high level structure of [CKP15], so we start by recalling this approach. The idea is to convert the VBB obfuscator $O^{\mathcal{I}}$ in the idealized model to an *approximate* VBB obfuscation $\widehat{O}$ in the plain model which gives the correct answer $C(x)$ with high (say, 9/10) probability over the choice of random input x and randomness of obfuscator. The final impossibility follows by applying the result of [BP13] which rules out approximate VBB in the plain model. Following [CKP15] our approximate VBB obfuscator $\widehat{O}$ in the plain model has the following high level structure.

1. Initial Obfuscation: Given a circuit C emulate the execution of the obfuscator $O^{\mathcal{I}}$ in the idealized model over input C to get circuit B (that performs in the idealized model).[1]

2. Learning phase: Emulate the execution of B over m random inputs for sufficiently large m.

3. Output B and the view z of the m executions above as the obfuscated code $\widehat{B} = (B, z)$.

4. Execute the obfuscated code $\widehat{B} = (B, z)$ on new random points using some form of lazy evaluation that only has access to the transcript z of the learning phase (and not the transcript of obfuscator O which is kept private). The exact solution here depends on the idealized model $\mathcal{I}$, but they all have the following form: if the answer to a new query could be "derived" from z use this answer, otherwise generate the answer from some simple distribution.

VBB property. As argued in [CKP15], the VBB property of $\widehat{O}$ follows from the VBB property of O and that the sequence of views z could indeed be sampled by any PPT holding B in the idealized model (by running B on m random inputs), and so it is simulatable (see Lemma 2.2).

Approximate correctness. The main challenge is to show the approximate correctness of the new obfuscation procedure in the plain model. The goal here is to show that if the learning phase is run for sufficiently large number of rounds, then its transcript z has enough information for emulating the next (actual) execution *consistently* with but *without* knowing the view of O. In the case that $\mathcal{I}$ is a random oracle [CKP15] showed that it is sufficient to bound the probability of the "bad" event E that the final execution of $\widehat{B} = (B, z)$ on a random input x asks any of the "private" queries of the obfuscator O which is not discovered during the learning phase. The work of [Pas15] studies graded encoding oracle model where the obfuscated code can perform arbitrary zero-test queries for low degree polynomials $p(\cdot)$ defined over the generated labels $s_1, \dots, s_k$. The oracle will return `true` if $p(s_1, \dots, s_k) = 0$ (in which case $p(\cdot)$ is called a zero polynomial) and returns `false` otherwise. Due to the algebraic structure of the field here, it is no longer enough to learn the heavy queries of the obfuscated code who might now ask its oracle query $p(\cdot)$ from some "flat" distribution while its answer is correlated with previous answers.

1.2.1 Generic Group Model: Proving Theorem 1.1

To describe the high level ideas of the proof of our Theorem 1.2 it is instructive to start with the proof of [Pas15] restricted to zero testing degree-1 polynomials and adapt it to the very close model

[1]The emulation here and in next steps would require the idealized model $\mathcal{I}$ to have an efficient "lazy evaluation" procedure. For example lazy evaluation for random oracles chooses a random answer (different from previous ones) given any new query.

of GGM for $\mathbb{Z}_p$ when p is a prime, since as noted in [Pas15] when it comes to zero-testing linear functions these two models are indeed very close.[2]

Case of $\mathbb{Z}_p$ for prime p [Pas15]. When we go to the generic group model we can ask addition and labeling queries as well. It can be shown that we do not need to generate any labels during obfuscation and they can be emulated using addition queries. Then, by induction, all the returned labels $t_1, \ldots, t_\ell$ for addition queries are linear combinations of $s_1, \ldots, s_k$ with integer coefficients.[3] and that is how we represent queries. Suppose we get an addition query $\mathbf{a} + \mathbf{b}$ and want to know the label of the group element determined by (the coefficients) $\mathbf{a} + \mathbf{b} = \mathbf{x}$. Suppose for a moment that we know s is the label for a vector of integers $\mathbf{c}$, and suppose we also know that the difference $\mathbf{x} - \mathbf{c}$ evaluates to zero. In this case, similarly to [CKP15], we can confidently return the label s as the answer to $\mathbf{a} + \mathbf{b}$. To use this idea, at any moment, let W be the space of all (zero) vectors $\boldsymbol{\alpha} - \boldsymbol{\beta}$ such that we have previously discovered same labels for $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$. Now to answer $\mathbf{a} + \mathbf{b} = \mathbf{x}$ we can go over all previously discovered labels ($\mathbf{c} \mapsto s$) and return s if $\mathbf{x} - \mathbf{c} \in \mathsf{span}(W)$, and return a random label otherwise. The approximate correctness follows from the following two points.

- **The rank argument.** First note that if $\mathbf{x} - \mathbf{c} \in \mathsf{span}(W)$ then the label for the vector $\mathbf{a} + \mathbf{b} = \mathbf{x}$ is indeed s. So we only need worry about cases where $\mathbf{x} - \mathbf{c} \notin \mathsf{span}(W)$ but $\mathbf{x} - \mathbf{c}$ is still zero. The rank argument of [Pas15] shows that this does not happen too often if we repeat the learning phase enough. The main idea is that if this "bad" event happens, it increases the rank of W, but this rank can increase only k times.

- **Gaussian elimination.** Finally note that the test $\mathbf{x} - \mathbf{c} \notin \mathsf{span}(W)$ can be implemented efficiently using Gaussian elimination when we work in $\mathbb{Z}_p$.

Case of general cyclic abelian groups. To extend the argument above to any cyclic abelian group $\mathbb{Z}_m$ (for possibly non-prime m) we need to consider the following two points:

- **Alternative notion for rank of W.** Unfortunately, when we move to the ring of $\mathbb{Z}_m$ for non-prime m it is no longer a field and we cannot simply talk about the rank of W (or equivalently the dimension of $\mathsf{span}(W)$) anymore.[4] More specifically, similarly to [Pas15], we need such (polynomial) bound to argue that during most of the learning phases the set $\mathsf{span}(W)$ does not grow. To resolve this issue we introduce an alternative notion which here we call $\widetilde{\operatorname{rank}}(W)$ that has the following three properties even when vectors $\mathbf{w} \in W$ are in $\mathbb{Z}_m^k$ (1) If $\mathbf{a} \in \mathsf{span}(W)$ then $\widetilde{\operatorname{rank}}(W) = \widetilde{\operatorname{rank}}(W \cup \{\mathbf{a}\})$, and (2) if $\mathbf{a} \notin \mathsf{span}(W)$ then $\widetilde{\operatorname{rank}}(W) + 1 \leq \widetilde{\operatorname{rank}}(W \cup \{\mathbf{a}\})$, and (3) $1 \leq \widetilde{\operatorname{rank}}(W) \leq k \cdot \log |\mathbb{Z}_m| = k \cdot \log m$. In particular in Lemma 3.15 we show that the quantity $\widetilde{\operatorname{rank}}(W) := \log |\mathsf{span}(W)|$ has these three properties. These properties together show that $\mathsf{span}(W)$ can only "jump up" $k \cdot \log m$ (or fewer) times during the learning phase, and that property is enough to be able to apply the argument of [Pas15] to show that sufficiently large number of learning phases will bound the error probability by arbitrary $1/\operatorname{poly}(n)$.

- **Solving system of linear equations over $\mathbb{Z}_m$.** Even though m is not necessarily prime, this can still be done using a generalized method for cyclic abelian groups [McC90].

[2]More formally, using the rank argument of [Pas15] it can be shown that for the purpose of obfuscation, the two models are equivalent up to arbitrary small $1/\operatorname{poly}(n)$ completeness error.

[3]Even though the summation is technically defined over the group elements, for simplicity we use the addition operation over the labels as well.

[4]Note that this is even the case for $\mathbb{Z}_q$ when q is a prime power (even though finite fields have prime power sizes.

Beyond cyclic groups. First note that to solve the Gaussian elimination algorithm for $\mathbb{Z}_m$, we first convert the integer vectors of W into some form of finite module by trivially interpreting the integer vectors of W as vectors in $\mathbb{Z}_m^k$. This "mapping" was also crucially used for bounding the quantity $\widetilde{\text{rank}}(W)$.

- **Mapping integers to general abelian G.** When we move to a general abelian group G we again need to have a similar mapping to map W into a "finite" module. Note that we do not know how to solve these questions using integer vectors in W efficiently. In Lemma 3.6 we show that a generalized mapping $\rho_G(\cdot)\colon \mathbb{Z} \mapsto G$ (generalizing the mapping $\rho_{\mathbb{Z}_m}(x) = (x \mod n)$ for $\mathbb{Z}_m$) exists for general abelian groups that has the same effect; Namely, without loss of generality we can first convert integer vectors in W to vectors in G^k and then work with the new W.

- **The alternative rank argument.** After applying the transformation above over W (to map it into a subset of G^k) we can again define and use the three rank-like properties of $\widetilde{\text{rank}}(\cdot)$ (instead of $\text{rank}(W)$) described above, but here for any finite abelian group G. In particular we use $\widetilde{\text{rank}}(W) := \log |\mathsf{span}_{\mathbb{Z}}(W)|$ where $\mathsf{span}_{\mathbb{Z}}(\cdot)$ is the module spanned by W using *integer* coefficients. Note that even though G is not a ring, multiplying integers with $x \in G$ is naturally defined (see Definition 3.5).

- **System of linear equations over finite abelian groups.** After the conversion step above, now we need to solve a system of linear equation $\mathbf{x}W = \mathbf{a}$ where elements of $W, \mathbf{a}$ are from G but we are still looking for *integer* vector solutions $\mathbf{x}$. After all, there is no multiplication defined for G. In Section A we show that solving this problem in polynomial time for general finite abelian groups could be reduced to the case of $G = \mathbb{Z}_m$, which is known [McC90].

1.2.2 Low-Degree Graded Encoding Model: Proving Theorem 1.2

To prove Theorem 1.3 for general finite rings, we show how to use the ideas developed for the case of general abelian generic groups discussed above and apply them to the framework of [Pas15] for low-degree graded encoding model as specified in Theorem 1.2.

Recall that here the goal is to detect the zero polynomials by checking their membership in the module $\mathsf{span}(W)$. Since here we deal with polynomials over a ring (or field) R the multiplication is indeed defined. Therefore, if we already know a set of zero polynomials W and want to judge whether $\mathbf{a}$ is also (the vector corresponding to) a zero polynomial, the more natural approach is to solve a system of linear equations $\mathbf{x}W = \mathbf{a}$ over ring R.

Searching for integer solutions, again. Unfortunately we are not aware of a polynomial time algorithm to solve $\mathbf{x} \cdot A = \mathbf{a}$ in general finite rings and as we mentioned above even special cases like $R = \mathbb{Z}_m$ are nontrivial [McC90]. Our idea is to try to reduce the problem back to the abelian *groups* by somehow eliminating the ring multiplication. Along this line, when we try to solve $\mathbf{x} \cdot W = \mathbf{a}$, we again restrict ourselves *only* to integer solutions. In other words, we do not multiply inside R anymore, yet we take advantage of the fact that the existence of integer solution to $\mathbf{x} \cdot A = \mathbf{a}$ is *still sufficient* to conclude $\mathbf{a}$ is a zero vector. As we mentioned above, we indeed know how to find integer solutions to such system of linear equations in polynomial time (see Theorem A.1).

Finally note that, we can again use our alternative rank notion of $\widetilde{\text{rank}}(W)$ to show that if we run the learning phase of the obfuscation (in plain model) m times the number of executions in which $\mathsf{span}_{\mathbb{Z}}(W)$ grows is at most $\mathrm{poly}(n)$ (in case of $O(1)$-degree polynomials). This means that we can still apply the high level structure of the arguments of [Pas15] for the case of finite rings *without* doing Gaussian elimination over rings (which seems to be a hard problem in general).

1.2.3 Random Trapdoor Permutation Model: Proving Theorem 1.3

Here we give the high-level intuition behind our impossibility result for the random TDP oracle.

Warm-up: Random Permutation Oracle. We start by first describing the proof for the simpler case of random permutation oracle. The transformation technique for the random oracle model can be easily adapted to work in the random permutation model as follows. For starters, assume that the random permutation is only provided on one input length k; namely $R\colon \{0,1\}^k \mapsto \{0,1\}^k$. If $k = O(\log n)$ where n is the security parameter, then it means that the whole oracle can be learned during the obfuscation and hardcoded in the obfuscated code, and so R cannot provide any advantage over the plain model.

On the other hand if $k = \omega(\log n)$ it means that the range of R is of super-polynomial size. As a result, the same exact procedure proposed in [CKP15] (that assumes R is a random oracle and shows how to securely compile out R from the construction) would also work if R is a random permutation oracle. The reason is that the whole transformation process asks $\mathrm{poly}(n)$ number of queries to R and, if the result of the transformation does not work when R is a random permutation, then the whole transformation gives a $\mathrm{poly}(n) = q$ query attack to distinguish between whether R is a random permutation or a random function. Such an attack cannot "succeed" with more than negligible probability when the domain of R has super-polynomial size $q^{\omega(1)}$ in the number of queries q.

In general, when the random permutation R is available in *all* input lengths, we can use a mixture of the above arguments. Namely, during the leaning phase whenever the obfuscated code asks any query from a domain of size $c\log(n)$ for a sufficiently large constant c, then we ask all of the queries in that domain and represent them in the new obfuscation (without the oracle). If the asked query is from a domain larger than $c\log(n)$, we pretend that this is a random oracle and use the same procedure as in [CKP15]. The above issues also exists in the trapdoor permutation and the generic group models and can be handled exactly the same way.

Random Trapdoor Permutation Model. When the ideal model is a random trapdoor permutation oracle $T = (G, F, F^{-1})$, it is not sufficient to only look at the event that an unlearned Q_O query was asked. Indeed, since the three algorithms are correlated, there is a possibility that the execution of B on the new random input might ask a new query that is *not* in Q_O, and yet still be inconsistent with some query in $Q_O \backslash Q_C$. For example, assume we have a query q of the form $F^{-1}[sk](y) = x$ that was asked during the emulation phase (and thus is in Q_O) but was missed in the learning phase (and thus is not in Q_C). It is possible that during the evaluation, the circuit B may ask a query q' of the form $F[pk](x)$ where $G(sk) = pk$, and since this is a new query, the plain-model circuit B' will answer with some random answer y'. Note that in this case, y' would be different from y with very high probability, and thus even though $q \neq q'$, they are together inconsistent with respect to oracle T. As a result, we should consider the events that may cause these conflicts.

To constrain the number of such bad events that need to analysed, we will "canonicalize" the obfuscated circuit B that is generated by O. That is to say, we will modify the behaviour of B

such that, for any query q that it asks, we would issue other queries that depend on q to ensure consistency with answers to previous queries in Q_C. Specifically, whenever B asks a new query q of the form $F^{-1}[sk](y)$, it will first issue a query $G(sk) = pk$. This way, if Q_C already contains a query $F[pk](x) = y$ for some x, we can set the answer of q to be x. Otherwise, we answer q with a new randomly chosen x. Using the canonicalization procedure, we provide a case-by-case analysis that the only bad event that we need to consider (besides hitting unlearned Q_O queries, which was already shown to be unlikely) is the event that $\widetilde{C'}$ asks a query of the form $F^{-1}[sk](y)$ for which query $F[pk](x)$ is in Q_O and $G(sk) = pk$.

2 Preliminaries

2.1 VBB Obfuscation

Definition 2.1 (Approximate VBB Obfuscation in Ideal Model [BGK+13, CKP15])**.** For a function $\epsilon(n) \in (0, 1)$, a PPT algorithm O is called an ϵ-approximate general purpose VBB obfuscator in the $\mathcal{I}$-ideal model if the following is satisfied:

- Approximate Functionality: For any circuit C of size n and input size m

  $$\Pr_{x \leftarrow \{0,1\}^m}[O^{\mathcal{I}}(C)(x) \neq C(x)] \leq \epsilon(n)$$

  where the probability is over the the choice of input x, the oracle $\mathcal{I}$, and the internal randomness of O.

- Virtual Black-Box: For every PPT adversary A, there exists a PPT simulator S and a negligible function $\sigma(n)$ such that for all $n \in \mathbb{N}$ and circuits $C \in \{0, 1\}^n$:

  $$\left|\Pr[A^{\mathcal{I}}(O^{\mathcal{I}}(C)) = 1] - \Pr[S^C(1^n) = 1]\right| \leq \sigma(n)$$

  where the probability is over the choice of $\mathcal{I}$ and the randomness of A, S, and O.

The following lemma is used in [CKP15] and we state it in an abstract form while ignoring the completeness of VBB security and only focusing on the security.

Lemma 2.2 (Preservation of VBB Security)**.** *Let O be a PPT algorithm in the $\mathcal{I}$-ideal model that satisfies VBB security, and let U be a PPT algorithm (in the $\mathcal{I}$-ideal model) that, given input $B = O^{\mathcal{I}}(C)$ for some circuit $C \in \{0, 1\}$ of size n, outputs (B, z) where z is some string. If there exists a plain-model PPT algorithm $\widehat{O}$ that, on input C, outputs (B', z') with distribution statistically close to (B, z) conditioned on C then $\widehat{O}$ also satisfies VBB security.*

Proof. For any fixed circuit C, $A^{\mathcal{I}}$ accepts as input $B = O^I(C)$ then executes $U^{\mathcal{I}}(B)$ to get (B, z). $A^{\mathcal{I}}$ will then run $\widehat{A}$ with input (B, z) then outputs whatever $\widehat{A}$ outputs. Given the behaviour of $A^{\mathcal{I}}$, we have that:

$$\Pr[A^{\mathcal{I}}(B) = 1] = \Pr[\widehat{A}(B, z) = 1] \tag{1}$$

Furthermore, if we let the statistical distance between (B, z) and $\widehat{O}(C) = (B', z')$ be at most $\epsilon(n)$, then we also have:

$$\left|\Pr[\widehat{A}(B', z') = 1] - \Pr[\widehat{A}(B, z) = 1]\right| \leq \epsilon(n) \tag{2}$$

Since O satisfies the security of VBB in the ideal model, we have that, there is a simulator S for the adversary $A^{\mathcal{I}}$ in the ideal model such that:

$$\left|\Pr[A^{\mathcal{I}}(B) = 1] - \Pr[S^C(1^n) = 1]\right| \leq \text{negl}(n) \tag{3}$$

Now let $\widehat{S}$ be a VBB simulator for $\widehat{A}$. Combining this with Equations 1 and 2, and given that $\epsilon(n)$ is a negligible function, we find that $\widehat{O}$ is also VBB-secure using the same simulator S. $\square$

2.2 Some Group Theory Facts

By $\mathbb{Z}$ we refer to the set of integers. By $\mathbb{Z}_n$ we refer to the additive (or maybe the ring) of integers modulo n. When G is an abelian group, we use $+$ to denote the operation in G. A semigroup $(G, \odot)$ consists of any set G and an associative binary operation $\odot$ over G.

Definition 2.3 (Direct Product of Semigroups)**.** For semi-groups $(G_1, \odot_1), \ldots, (G_k, \odot_1)$, by the direct product semi-group $(G, \odot) = (G_1 \times \cdots \times G_k, \odot_1 \times \cdots \times \odot_k)$ we refer to the group in which for $g = (g_1, \ldots, g_k) \in G, h = (h_1, \ldots, h_k) \in G$ we define $g \odot h = (g_1 \odot_1 h_1, \ldots, g_k \odot_k h_1)$. If G_i's are groups, their direct product is also a group.

The following is the fundamental theorem of finitely generated abelian groups restricted to case of finite abelian groups.

Theorem 2.4 (Characterization of Finite Abelian Groups)**.** *Any finite abelian group G is isomorphic to some direct product group $\mathbb{Z}_{p_1^{\alpha_1}} \times \cdots \times \mathbb{Z}_{p_d^{\alpha_d}}$ in which p_i's are (not necessarily distinct) primes and $\mathbb{Z}_{p_i^{\alpha_i}}$ is the additive group mod $p_i^{\alpha_i}$.*

2.3 The Generic Group Model

Definition 2.5 (Generic Group Model [Sho97])**.** Let $(G, \odot)$ be any group of size N and let S be any set of size at least N. The *generic* group oracle $\mathcal{I}[G \mapsto S]$ (or simply $\mathcal{I}$) is as follows. At first an injective random function $\sigma \colon G \mapsto S$ is chosen, and two type of queries are answered as follows.

- **Type 1: Labeling Queries.** Given $g \in G$ oracle returns $\sigma(g)$.
- **Type 2: Addition Queries.** Given y_1, y_2, if there exists x_1, x_2 such that $\sigma(x_1) = y_1$ and $\sigma(x_2) = y_2$, it returns $\sigma(x_1 \odot x_2)$. Otherwise it returns $\perp$.

Definition 2.6. [Generic Algorithms in Generic Group Model] Let $A^{\mathcal{I}}$ be an algorithm (or a set of interactive algorithms $A = \{A_1, A_2, \ldots\}$) accessing the generic group oracle $\mathcal{I}[G \mapsto S]$. We call $A^{\mathcal{I}}$ generic if it never asks any query (of the second type) that is answered as $\perp$. Namely, only queries are asked for which the labels are previously obtained.

Remark 2.7 (Family of Groups)**.** A more general definition allows generic oracle access to a *family* of groups $\{G_1, G_2, \ldots\}$ in which the oracle access to each group is provided separately when the index i of G_i is also specified as part of the query and the size of the group G_i is known the parties. Our negative result of Section 3 directly extends to this model as well. We use the above "single-group" definition for sake of simplicity.

Remark 2.8 (Stateful vs Stateless Oracles and the Multi-Party Setting)**.** Note that in the above definition we used a *stateless* oracle to define the generic group oracle, and we separated the generic nature of the oracle itself from *how* it is used by an algorithm $A^{\mathcal{I}}$. In previous work (e.g., Shoup's original definition [Sho97]) *a stateful* oracle is used such that: it will answer addition queries *only* if the two labels are already *obtained* before.[5]

Note that for "one party" settings in which $A^{\mathcal{I}}$ is a single algorithm, $A^{\mathcal{I}}$ "knows" the labels that it has already obtained from the oracle $\mathcal{I}$, and so w.l.o.g. $A^{\mathcal{I}}$ would never ask any addition queries unless it has previously obtained the labels itself. However, in the multi-party setting, a party might not know the set of labels obtained by other parties. A stateful oracle in this case might reveal some information about other parties' oracle queries if the oracle does *not* answer a query (y_1, y_1) (by returning $\perp$) just because no one has obtained the labels y_1, y_2 so far.

Remark 2.9 (Equivalence of Two Models for Sparse Encodings)**.** If the encoding of G is sparse in the sense that $|S|/|G| = n^{\omega(1)}$ where n is the security parameter, then the probability that any party could query a correct label before it being returned by oracle through a labeling (type 1) query is indeed negligible. So in this case any algorithm (or set of interactive algorithms) $A^{\mathcal{I}}$ would have a behavior that is statistically close to a generic algorithm that would never ask a label in an addition query unless that label is previously obtained from the oracle. Therefore, if $|S|/|G| = n^{\omega(1)}$, we can consider $A^{\mathcal{I}}$ to be an *arbitrary* algorithm (or set of interactive algorithms) in the generic group model $\mathcal{I}$. The execution of A would be statistically close to a "generic execution" in which $A^{\mathcal{I}}$ never asks any label before obtaining it.

2.4 Ideal Graded Encoding Model

We adapt the following definition from [Pas15] restricted to the degree-d polynomials. For simplicity, as in [Pas15] we also restrict ourselves to the setting in which only the obfuscator generates labels and the obfuscated code only does zero tests, but the proof directly extends to the more general setting of [BGK+14, BR14] We also use only one finite ring R in the oracle (whose size could in fact depend on the security parameter) but our impossibility result extends to any sequence of finite rings as well.

Definition 2.10 (Degree-d Ideal Graded Encoding Model)**.** The oracle $\mathcal{M}^d_R = (\mathrm{enc}, \mathrm{zero})$ is stateful and is parameterized by a ring R and a degree d and works in two phases. For each l the oracle $\mathrm{enc}(\cdot, l)$ is a random injective function from the ring R to the set of labels $S = \{0,1\}^{3\cdot|R|}$.

1. Initialization phase: In this phase the oracle answers $\mathrm{enc}(v, l)$ queries and for each query it stores (v, l, h) in a list $\mathcal{L}_O$.
2. Zero testing phase: Suppose $p(\cdot)$ is a polynomial whose coefficients are explicitly represented in R and its monomials are represented with labels $h_1, \dots, h_m$ obtained through $\mathrm{enc}(\cdot,\cdot)$ oracle in phase 1. Given any such query $p(\cdot)$ the oracle answers as follows:
   (a) If any h_i is not in $\mathcal{L}_O$ (i.e., it is not obtained in phase 1) return false.
   (b) If the degree of $p(\cdot)$ is more than d then return false.
   (c) Let $(v_i, l_i, h_i) \in \mathcal{L}_O$. If $p(v_1, \dots, v_m) = 0$ return true; otherwise return false.

[5]So the oracle might return $\perp$ even if the two labels are in the range of $\sigma(G)$.

Remark 2.11. Remarks 2.8 and 2.9 regarding the stateful vs stateless oracles and the sparsity of the encoding in the context of generic group model apply to the graded encoding model as well. Therefore, as long as the encoding is sparse (which is the case in the definition above whenever $|R|$ is of size $n^{\omega(1)}$) the probability of obtaining any valid label $h = \mathrm{enc}(v, l)$ through any polynomial time algorithm without it being obtained from the oracle previously (by the same party or another party) becomes negligible, and so the model remains essentially equivalent (up to negligible error) even if the oracle does not keep track of which labels are obtained previously through $\mathcal{L}_O$.

2.5 Random Trapdoor Permutation Oracle

Let Π_n be the set of all permutations mapping $\{0,1\}^n$ to $\{0,1\}^n$.

Definition 2.12 (Random Trapdoor Permutation)**.** For any security parameter n, a random *trapdoor permutation* (TDP) oracle T_n consists of three subroutines (G, F, F^{-1}) defined as follows:

- $G(sk)$: is is chosen at random from Π_n and maps any trapdoor $sk \in \{0,1\}^n$ to a public index $pk \in \{0,1\}^n$.
- $F[pk](x)$: For any fixed public index pk, $F[pk](\cdot)$ is a random permutation over $\{0,1\}^n$.
- $F^{-1}[sk](y)$: For any fixed trapdoor sk such that $G(sk) = pk$, $F^{-1}[sk](\cdot)$ is the inverse permutation of $F[pk](\cdot)$, namely $F^{-1}[sk](F[pk](x)) = x$.

Definition 2.13 (Random Hierarchical Trapdoor Permutation)**.** For any security parameter n and $l = \mathrm{poly}(n)$, an l-level random *hierarchical trapdoor permutation* (HTDP) oracle T_n^l consists of $2l+3$ subroutines $(\{G_i\}_{i=1}^{l+1}, \{H_i^{-1}\}_{i=0}^{l+1})$ defined as follows:

- $H_i^{-1}[\mathrm{ID}_{i-2}, id_{i-1}](td_i)$: A permutation, indexed by identity vector $\mathrm{ID}_{i-2} = (id_0, ..., id_{i-2})$ and id_{i-1}, that accepts as input an i-level trapdoor $td_i \in \{0,1\}^n$ and outputs a randomly chosen identity $id_i \in \{0,1\}^n$.[6]
- $G_i[\mathrm{ID}_{i-2}, td_{i-1}](id_i)$: A permutation, indexed by identity vector $\mathrm{ID}_{i-2} = (id_0, ..., id_{i-2})$ and td_{i-1} that, given the identity $id_i \in \{0,1\}^n$, outputs the corresponding trapdoor $td_i \in \{0,1\}^n$.

3 Impossibility of VBB in Generic Abelian Group Model

In this section we will state and prove Theorem 1.1 formally. In light of Remarks 2.8 and 2.9 above, for simplicity of the exposition we will always assume that the encoding is sparse $|S|/|G| = n^{\omega(1)}$ and so all the generic group model are automatically (statistically close to being) generic.

Theorem 3.1. *Let G be any abelian group of size at most $2^{\mathrm{poly}(n)}$. Let O be an obfuscator in the generic group model $\mathcal{I}[G \mapsto S]$ where the obfuscation of any circuit followed by execution of the obfuscated code (jointly) form a generic algorithm. If O is an ϵ-approximate VBB obfuscator in the generic group model (G, S) for poly-size circuits, then for any $\delta = 1/\mathrm{poly}(n)$ there exists an $(\epsilon + \delta)$-approximate obfuscator $\widehat{O}$ for poly-size circuits in the plain model.*

[6]This permutation is functionally equivalent to the inverse of G_i but, since it is indexed differently, we present a more well-defined notation to distinguish it from G_i^{-1}, which is indexed with $(\mathrm{ID}_{i-2}, td_{i-1})$.

Remark 3.2 (Size of G)**.** Note that if a poly(n)-time algorithm accesses (the labels of the elements of) some group G, it implicitly means that G is at most of $\exp(n)$ size so that its elements could be names with poly(n) bit strings. We chose, however, to explicitly mention this size requirement $|G| \leq 2^{\text{poly}(n)}$ since this upper bound plays a crucial role in our proof for general abelian groups compared to the special case of finite fields.

Remark 3.3 (Sparse Encodings)**.** If we assume a sparse encoding i.e., $|S|/|G| = n^{\omega(1)}$ (as e.g., is the case in [Pas15] and almost all prior work in generic group model) in Theorem 3.1 we no longer need to explicitly assume that the obfuscation followed by execution of obfuscated code are in generic form; see Remark 2.9.

Since [BP13] showed that (assuming TDPs) there is no $(1/2 - 1/\text{poly})$-approximate VBB obfuscator in the plain-model for general circuits, the following corollary is obtained by taking $\delta = \epsilon/2$.

Corollary 3.4. *If TDPs exist, then there exists no $(1/2 - \epsilon)$-approximate VBB obfuscator O for general circuits in the generic group model for any $\epsilon = 1/\text{poly}(n)$, any finite abelian group G and any label set S of sufficiently large size $|S|/|G| = n^{\omega(1)}$. The result would hold for labeling sets S of arbitrary size if the execution of the obfuscator O followed by the execution of the obfuscated circuit $O(C)$ form a generic algorithm.*

Before proving Theorem 3.1 we need some basic group theoretic definitions and tools.

Definition 3.5 (Integer vs in-group multiplication for abelian groups)**.** For integer $a \in \mathbb{Z}$ and $g \in G$ where G is any finite abelian group by $a \cdot g$ we mean adding g by itself $|a|$ times and negating it if $a < 0$. Now let $g, h \in G$ both be from abelian group G and let $G = \mathbb{Z}_{p_1^{\alpha_1}} \times \cdots \times \mathbb{Z}_{p_d^{\alpha_d}}$ where p_i's are primes. If not specified otherwise, by $g \cdot h$ we mean the multiplication of g, h in G interpreted as the multiplicative semigroup that is the direct product of the *multiplicative* semigroups of $\mathbb{Z}_{p_i^{\alpha_i}}$'s for $i \in [d]$ (where the multiplications in $\mathbb{Z}_{p_i^{\alpha_i}}$ are done mod $\mathbb{Z}_{p_i^{\alpha_i}}$).

Lemma 3.6 (Mapping integers to abelian groups)**.** *Let $G = \mathbb{Z}_{p_1^{\alpha_1}} \times \cdots \times \mathbb{Z}_{p_d^{\alpha_d}}$. Define $\rho_G \colon \mathbb{Z} \mapsto G$ as $\rho_G(a) = (a_1, \ldots, a_d) \in G$ where $a_i = a \mod p_i^{\alpha_i} \in \mathbb{Z}_{p_i^{\alpha_i}}$. Also for $\mathbf{a} = (a_1, \ldots, a_k) \in \mathbb{Z}^k$ define $\rho_G(\mathbf{a}) = (\rho_G(a_1), \ldots, \rho_G(a_k))$. Then for any $a \in \mathbb{Z}$ and $g \in G = \mathbb{Z}_{p_1^{\alpha_1}} \times \cdots \times \mathbb{Z}_{p_d^{\alpha_d}}$ it still holds that $a \cdot g = \rho_G(a) \cdot g$ where the first multiplication is done according to Definition 3.5, and the second multiplication is done in G according to Definition 3.5. More generally, if $\mathbf{a} = (a_1, \ldots, a_k) \in \mathbb{Z}^k$, and $\mathbf{g} = (g_1, \ldots, g_k) \in G$, then $\sum_{i \in [k]} a_i g_i = \langle \mathbf{a}, \mathbf{g} \rangle = \langle \rho_G(\mathbf{a}), \mathbf{g} \rangle$.*

Remark 3.7 (General abelian vs $\mathbb{Z}_n$.)**.** Note that when $G = \mathbb{Z}_n$ is cyclic, the mapping $\rho_G \colon \mathbb{Z} \mapsto G$ of Lemma 3.6 will be equivalent to simply mapping every $a \in \mathbb{Z}$ to $(a \mod n) \in G$. Therefore, Definition 3.5 generalizes the notion of $\mathbb{Z}_n$ a a ring to general abelian groups, since the multiplication $x \cdot y \mod n$ in $\mathbb{Z}_n$ is the same as a multiplication in which x is interpreted from $\mathbb{Z}$ (as in Definition 3.5) which is equivalent to doing the multiplication inside G according to by Lemma 3.6.

In the rest of this section now prove Theorem 3.1. We will first describe the algorithm of the obfuscator in the plain model, and then will analyze its completeness and soundness.

Notation and w.l.o.g. assumptions. Using Theorem 2.4 w.l.o.g. we assume that our abelian group G is isomorphic to the additive direct product group $\mathbb{Z}_{p_1^{\alpha_1}} \times \cdots \times \mathbb{Z}_{p_d^{\alpha_d}}$ where p_i's are prime.

Let $e_i \in G$ be the vector that is 1 in the i'th coordinate and zero elsewhere. Note that $\{e_1, \dots, e_k\}$ generate G. We can always assume that the first d labels obtained by O are the labels of $e_1, \dots, e_d$ and these labels are explicitly passed to the obfuscated circuit $B = O(C)$. Let $k = \operatorname{poly}(n)$ be an upper bound on the running time of the obfuscator O for input C which in turn upper bounds the number of labels obtained during the obfuscation (including the the d labels for $e_1, \dots, e_d$). We also assume w.l.o.g. that the obfuscated code never asks any type one (i.e., labeling) oracle queries since it can use the label for $e_1, \dots, e_d$ to obtain labels of any arbitrary $g = a_1 e_1 + \cdots + a_d e_d$ using a polynomial number of addition (i.e., type two) oracle queries. For $\sigma(g) = s$, $a \in \mathbb{Z}$, and $t = \sigma(a \cdot g)$ we abuse the notation and denote $a \cdot s = t$.

3.1 The Construction

Even though the output of the obfuscator is always an actual circuit, we find it easier to first describe how the obfuscator $\widehat{O}$ generates some "data" $\widehat{B}$, and then we will describe how to use $\widehat{B}$ to execute the new obfuscated circuit in the plain model. For simply we also use $\widehat{B}$ to denote the obfuscated circuit.

3.1.1 How to Obfuscate

The new obfuscator $\widehat{O}$. The new obfuscator $\widehat{O}$ uses lazy evaluation to simulate the labeling $\sigma(\cdot)$ oracle. For this goal, it holds a set Q_σ of the generated labels. For any new labeling query $g \in G$ if $\sigma(g) = s$ is already generated it returns s. Otherwise it chooses an unused label s from S uniformly at random and adds the mapping $(g \to s)$ to Q_σ and returns s. For an addition query (s_1, s_2) it first finds g_1, g_2 such that $\sigma(g_1) = s_1$ an $\sigma(g_2) = s_2$ (which exist since the algorithm that calls the oracle is in generic form) and gets $g = g_1 + g_2$. Now $\widehat{O}$ proceeds as if g is asked as a labeling query and returns the answer. The exact steps of $\widehat{O}$ are as follows.

1. *Emulating obfuscation.* $\widehat{O}$ emulates the randomized obfuscation $O^{\mathcal{I}[G \mapsto S]}(C)$ to get circuit B.
2. *Learning phase 1 (heavy queries)*: Set $Q_B = \varnothing$. For $i \in [d]$ let $t_i = \sigma(e_i)$ be the label of $e_i \in G$ which is explicitly passed to B by the obfuscator $O(C)$ and $T = (t_1, \dots, t_d)$ at the beginning. The length of the sequence T would increase during the steps below but will never exceed k. Choose m at random from $\ell = [\lceil 3 \cdot k \cdot \log(|G|)/\delta \rceil]$. For $i = 1, \dots, m$ do the following:
    - Choose x_i as a random input for B. Emulate the execution of B on x_i using the (growing) set Q_σ of partial labeling for the lazy evaluation of labels. Note that as we said above, w.l.o.g. B only asks addition (i.e., type two) oracle queries. Suppose B (executed on x_i) needs to answer an addition query (s_1, s_2). If either of the labels $u = s_1$ or $u = s_2$ is *not* already obtained during the learning phase 1 (which means it was obtained during the initial obfuscation phase) append u to the sequence T of discovered labels by $T := (T, u)$. Using induction, it can be shown that for any addition query asked during learning phase 1, at the time of being asked, we would know that the answer to this query will be of the form $\sum_{i \in [k]} a_i \cdot t_i$ for integers a_i. Before seeing why this is the case, let $\mathbf{a}_i = (a_{i,1}, \dots, a_{i,k})$ be the vector of integer coefficients (of the labels $t_1, t_2, \dots$) for the answer s that is returned to the i'th query of learning phase 1. We add $(\mathbf{a}_i \to s)$ to Q_B for the returned label. To see why such vectors exist, let (s_1, s_2) be an addition query asked during this phase, and let $s \in \{s_1, s_2\}$. If the label s is obtained previously

during learning phase 1, then the linear form $s = \sum_{i \in [k]} a_i \cdot t_i$ is already stored in Q_B. On the other hand, if s is a new label discovered during an addition (i.e., type two) oracle query which just made $T = (t_1, \dots, t_{j-1}, t_j = s)$ have length j, then $s = a_i \cdot t_i$ for $a_j = 1$. Finally, if the linear forms for both of (s_1, s_2) in an addition oracle query are known, the linear form for the answer s to this query would be the summation of these vectors.[7]

3. *Learning phase 2 (zero vectors)*: This step does not involve executing B anymore and only generates a set $W = W(Q_B) \subseteq G^k$ of polynomial size. At the beginning of this learning phase let $W = \varnothing$. Then for all $(\mathbf{a}_1 \to s_1) \in Q_B$ and $(\mathbf{a}_2 \to s_2) \in Q_B$, if $s_1 = s_2$, let $\mathbf{a} = \mathbf{a}_1 - \mathbf{a}_2$, and add $\rho_G(\mathbf{a})$ to W where $\rho_G(\mathbf{a})$ is defined in Lemma 3.6.

4. The output of the obfuscation algorithm will be $\widehat{B} = (B, Q_B, W, T, r)$ where T is the current sequence of discovered labels $(t_1, t_2, \dots)$ as described in Lemma 3.6, and r is a sufficiently large sequence of random bits that will be used as needed when we run the obfuscated code $\widehat{B} = (B, Q_B, W, T, r)$ in the plain model.[8]

3.1.2 How to Execute

In this section we describe how to execute $\widehat{B}$ on an input x using (B, Q_B, W, T, r).[9] Before describing how to execute the obfuscated code, we need to define the following algebraic problem.

Definition 3.8. [Integer Solutions to Linear Equations over Abelian Groups (iLEAG)] Let G be an abelian group. Suppose we are given G (e.g., by describing its decomposition factors according to Theorem 2.4) an $n \times k$ matrix A with components from G and a vector $\mathbf{b} \in G^k$. We want to find an *integer* vector $\mathbf{x} \in \mathbb{Z}^n$ such that $\mathbf{x}A = \mathbf{b}$.

Remark 3.9 (Integer vs. Ring Solutions)**.** Suppose instead of searching for an integer vector solution $\mathbf{x} \in \mathbb{Z}^n$ we would ask to find $\mathbf{x} \in G^n$ and define multiplication in G according to Definition 3.5 and call this problem G-LEAG. Then any solution to iLEAG can be directly turned into a solution for G-LEAG by mapping any integer coordinate x_i of $\mathbf{x}$ into G by mapping $\rho_G(x_i)$ of Lemma 3.6. The converse is true also for $G = \mathbb{Z}_n$, since any $g \in \mathbb{Z}_n$ is also in $\mathbb{Z}$ and it holds that $\rho_G(g) = g \in G$. However, the converse is not true in general for general abelian groups, since there could be members of G that are not in the range of $\rho_G(\mathbb{Z})$. For example let $G = \mathbb{Z}_{p^2} \times \mathbb{Z}_p$ for prime $p > 2$ and let $g = (2, 1)$. Note that there is no integer a such that $a \mod p^2 = 2$ but $a \mod p = 1$.

Executing $\widehat{B}$. The execution of $\widehat{B} = (B, Q_B, W, T, r)$ on x will be done *identically* to to the "next" execution during the learning phase 1 of the obfuscation (as if x is the $(m+1)$'st execution of this learning phase) and even the sets $Q_B, W = W(Q_B)$ will grow as the execution proceeds,

[7] Note that although the sequence T grows as we proceed in learning phase 1, we already now that this sequence will not have length more than d since all of these labels that are discovered while executing the obfuscated code has to be generated by the obfuscator, due to the assumption that the sequential execution of the obfuscator followed by the obfuscated code is in generic form. Therefore we can always consider $\mathbf{a}_i$ to be of dimension k.

[8] Note that even though $W(Q_B)$ could always be derived from Q_B, and even T could be derived from an *ordered* variant of Q_B (in which the order in which Q_B has grown is preserved) we still choose to explicitly represent these elements in the obfuscated $\widehat{B}$ to ease the description of the execution of $\widehat{B}$.

[9] Note that we do not have access to the set Q_σ that was used for consistent lazy evaluation of $\sigma(\cdot)$.

with the *only* difference described as follows.[10] Suppose we want to answer an addition (i.e., type two) oracle query (s_1, s_2) where for $b = \{1, 2\}$ we inductively know that $s_b = \sum_{i \in [k]} a_{b,i} \cdot t_i$. For $b = \{1, 2\}$ let $\mathbf{a}_b = (a_{b,1}, \dots, a_{b,k})$ and let $\mathbf{a} = \mathbf{a}_1 + \mathbf{a}_2$.

- Do the following for all $(\mathbf{b} \to s) \in Q_B$. Let $\mathbf{c} = \mathbf{a} - \mathbf{b}$ and let $\bar{\mathbf{c}} = \rho_G(\mathbf{c}) \in G^k$ as defined in Lemma 3.6. Let A be a matrix whose rows consists of all vectors in W. Run the polynomial time algorithm of Theorem A.1 to see if there is any integer solution $\mathbf{v}$ for $\mathbf{v}A = \bar{\mathbf{c}}$ as an instance of the iLEAG problem defined in Definition 3.8. If an integer solution $\mathbf{v}$ exists, then return s as the result (recall $(\mathbf{b} \to s) \in Q_B$), break the loop, and continue the execution of $\widehat{B}$. If the loop ended and no such $(\mathbf{b} \to s) \in Q_B$ was found, choose a random label s not in Q_B as the answer, add $(\mathbf{a} \to s)$ to Q_B and continue.

3.2 Completeness and the Soundness

In this section we prove the completeness and soundness of the construction of Section 3.1.

Size of S. In the analysis below, we will assume w.l.o.g. that the set of labels S has superpolynomial size $|S| = n^{\omega(1)}$. This would immediately hold if the labeing of G is sparse, since it would mean even $|S|/|G| \geq n^{\omega(1)}$. Even if the labeling is not sparse, we will show that w.l.o.g. we can assume that G itself has super-polynomial size (which means that S will be so too). That is because otherwise all the labels in G can be obtained by the obfuscator, the obfuscated code, and the adversary and we will be back to the plain model. More formally, for this case Theorem 3.1 could be proved through a trivial construction in which the new obfuscator simply generates all the labels of G and plant all of them in the obfuscated code and they will be used by the obfuscated algorithm. More precisely, and when the size of G (as a function of security parameter n) is neither of polynomial size $|G| = n^{O(1)}$ nor super-polynomial size $|G| = n^{\omega(1)}$ we can still choose a sufficiently large polynomial $\gamma(n)$ and generate all labels of G when $|G| < \gamma(n)$, and otherwise use the obfuscation of Section 3.1.

Completeness: approximate functionality. Here we prove the the following claim.

Claim 3.10. *Let $\widehat{B} = (B, Q_B, W, T, r)$ be the output of the obfuscator $\widehat{O}$ given input circuit C with input length α. If we run $\widehat{B}$ over a* random *input according to the algorithm described in Section 3.1, then it holds that*

$$\Pr_{x \leftarrow \{0,1\}^\alpha, \widehat{B} \leftarrow \widehat{O}(C)} \left[\widehat{B}(x) \neq C(x)\right] \leq \Pr_{x \leftarrow \{0,1\}^\alpha, B \leftarrow O^{\mathcal{I}[G \mapsto S]}(C)} \left[B^{\mathcal{I}[G \mapsto S]}(x) \neq C(x)\right] + \delta$$

over the randomness of $\mathcal{I}[G \mapsto S]$, random choice of x and the randomness of the obfuscators.

Proof. As a mental experiment, suppose we let the learning phase 1 always runs for exactly $\ell + 1 = 1 + [\lceil 3 \cdot k \cdot \log(|G|)/\delta \rceil]$ rounds but only derive the components $(Q_B, W(Q_B), T)$ based on the first m executions. Now, let x_i be the random input used in the i'th execution and y_i be the output of the i'th emulation execution the learning phase 1. Sine all the executions of the learning phase 1 are perfect simulations, for every $i \in [\ell]$, and in particular $i = m$, it holds that

$$\Pr[B^{\mathcal{I}[G \mapsto S]}(x) \neq C(x)] = \Pr[y_i \neq C(x)]$$

[10]We even allow new labels t_i to be discovered during this execution to be appended to T, even though that would indirectly lead to an abort!

where probability is over the choice of inputs x, x_i as well as all other randomness in the system. Thus, to prove claim 3.10 it will suffice to prove that

$$|\Pr[y_i \neq C(x)] - \Pr[\widehat{B}(x_i) \neq C(x)]| < \delta.$$

We will indeed do so by bounding the statistical distance between the execution of $\widehat{B}$ vs the $m+1$'st execution of the learning phase 1 over the same input x_i. Here we will rely on the fact that m is chosen at random from $[\ell]$.

Claim 3.11. *For random $[\ell]$ the statistical distance between the $m+1$'st execution of the learning phase 1 (which we call B') and the execution of $\widehat{B}$ over the* same *input x_i is at most $2\delta/3 + \operatorname{negl}(n)$.*

To prove the above claim we will define 3 type of bad events over a joint execution of $B' = B_{m+1}$ and $\widehat{B}$ when they are done concurrently and using the same random tapes (and even the input x_i). We will then show that (1) as long as these bad events do not happen the two executions proceed identically, and (2) the total probability of these bad events is at most $2\delta/3 + \operatorname{negl}(n)$. In the following we suppose that the executions of B' and $\widehat{B}$ (over the same random input) has proceeded identically so far. Suppose we want to answer an addition (i.e., type two) oracle query (s_1, s_2) where for $b = \{1, 2\}$ we inductively know that $s_b = \sum_{i \in [k]} a_{b,i} \cdot t_i$. Several things could happen:

- If the execution of $\widehat{B}$ finds $(\mathbf{b} \to s) \in Q_B$ such that when we take $\mathbf{c} = \mathbf{a} - \mathbf{b}$ and let $\overline{\mathbf{c}} = \rho_G(\mathbf{c}) \in G^k$ and let A be a matrix whose rows are vectors in (the current) W, there is an integer solution $\mathbf{v}$ to the iLEAG instance $\mathbf{v}A = \overline{\mathbf{c}}$. If this happens the execution of $\widehat{B}$ will use $\mathbf{b}$ as the answer. We claim that this is the "correct" answer as B' would also use the same answer. This is because by the definition of W and Lemma 3.6 for all $\mathbf{w} \in W$ it holds that $\mathbf{w} = (w_1, \dots, w_k)$ is a "zero vector in G^k" in the sense that summing the (currently discovered labels in) T with coefficients $w_1, \dots, w_k$ (and multiplication defined according to Definition 3.5) will be zero. As a result, $\mathbf{v}A = \overline{\mathbf{c}}$ which is a linear combination of vectors in W with integer coefficients will also be a zero vector. Finally, by another application of Lemma 3.6 it holds that $(c_1, \dots, c_k) = \mathbf{c} = \mathbf{a} - \mathbf{b}$ is a "zero vector in $\mathbb{Z}^k$ in the sense that summing the (currently discovered labels in) T with *integer* coefficients $c_1, \dots, c_k$ (and multiplication defined according to Definition 3.5) will also be zero. Therefore the answer to the query defined by vector $\mathbf{a}$ is equal to the answer defined by vector $\mathbf{b}$ which is the label s.

- If the above does not happen (and no such $(\mathbf{b} \to s) \in Q_B$ is found) then either of the following happens. Suppose the answer returned for (s_1, s_2) in execution of B' is s':

  - **Bad event E_1:** s' is equal to one of the labels in Q_B. Note that in this case the executions will diverge because $\widehat{B}$ will choose a random label.
  - **Bad event E_2:** s' is equal to one of the labels discovered in the emulation of $O^{\mathcal{I}}(C)$ (but not present in the current Q_B).
  - **Bad event E_3:** s' is a new label, but the label chosen by $\widehat{B}$ is one of the labels used in the emulation of $O^{\mathcal{I}}(C)$. (Note that in this case the execution of $\widehat{B}$ will not use any previously used labels in Q_B.

It is easy to see that as long as none of the events E_1, E_2, E_3 happen, the execution of B' and $\widehat{B}$ proceeds statistically the same. Therefore, to prove Claim 3.11 and so Claim 3.10 it is sufficient to bound the probability of the events E_1, E_2, E_3 as we do below.

Claim 3.12. $\Pr[E_3] < \text{negl}(n)$.

Proof. This is because (as we described at the beginning of this subsection above) the size of S is $n^{\omega(1)}$ but the number of labels discovered in the obfuscation phase is at most $k = \text{poly}(n)$. Therefore the probability that a random label from S after excluding labels in Q_B (which is also of polynomial size) hits one of at most k possible labels is $\leq k/(|S| - |Q_B|) = \text{negl}(n)$. Therefore, the probability that E_3 happens for any of the oracle quries in the execution of $\widehat{B}$ is also $\text{negl}(n)$. $\square$

Claim 3.13. $\Pr[E_2] < \delta/(3 \log |G|) < \delta/3$.

Proof. We will prove this claim using the randomness of $m \in [\ell]$. Note that every time that a label u is discovered in learning phase 1, this label u cannot be discovered "again", since it will be in Q_B from now on. Therefore, the number of possible indexes of $i \in [\ell]$ such that during the i'th execution of the learning phase 1 we discover a label out of Q_B is at most k. Therefore, over the randomness of $m \leftarrow [\ell]$ the probability that the $m+1$'st execution discovers any new labels (generated in the obfuscation phase) is at most $k/\ell \leq \delta/(3 \log |G|)$. $\square$

Claim 3.14. $\Pr[E_1] < \delta/3$.

Proof. Call $i \in [\ell]$ a bad index, if event E_3 happens conditioned on $m = i$ during the execution of B' (which is the $(m+1)$'s execution of learning phase 1). Whenever E_3 happens at any moment, it means that the vector $\overline{\mathbf{c}}$ is not currently in $W(Q_B)$, *but* it *will* be added W just after this query is made. We will show (Lemma 3.15 below) that the size of $\mathsf{span}_{\mathbb{Z}}(W)$ will at least double after this oracle query for some set $\mathsf{span}_{\mathbb{Z}}(W)$ that depends on W and that $\mathsf{span}_{\mathbb{Z}}(W) \subseteq G^k$, and so $|\mathsf{span}_{\mathbb{Z}}(W)| \leq |G|^k$. As a result the number of bad indexes i will be at most $\log |G|^k = k \log |G|$. Therefore, over the randomness of $m \in [\ell]$ the probability that $m+1$ is a bad index is at most $k \log |G| / \ell \leq \delta/3$ $\square$

Lemma 3.15. *Let $W \subseteq G^k$ for some abelian group G. Let $\mathsf{span}_{\mathbb{Z}}(W) = \{\sum_{\mathbf{w} \in W} a_{\mathbf{w}} \mathbf{w} \mid a_{\mathbf{w}} \in \mathbb{Z}\}$ be the module spanned by W using integer coefficients. If $\overline{\mathbf{c}} \notin \mathsf{span}_{\mathbb{Z}}(W)$, then it holds that*

$$|\mathsf{span}_{\mathbb{Z}}(W \cup \{\overline{\mathbf{c}}\})| \geq 2 \cdot |\mathsf{span}_{\mathbb{Z}}(W)|.$$

Proof. Let $A = \mathsf{span}_{\mathbb{Z}}(W)$ and let $B = \{\overline{\mathbf{c}} + \mathbf{w} \mid \mathbf{w} \in \mathsf{span}_{\mathbb{Z}}(W)\}$ be A shifted by $\overline{\mathbf{c}}$. It holds that $|A| = |B|$ and $A \cup B \subset \mathsf{span}_{\mathbb{Z}}(W \cup \{\overline{\mathbf{c}}\})$. It also holds that $A \cap B = \varnothing$, because otherwise then we would have: $\exists i, j : \mathbf{w} + \overline{\mathbf{c}} = \mathbf{w}'$ for $\mathbf{w}, \mathbf{w}' \in \mathsf{span}_{\mathbb{Z}}(W)$ which would mean $\overline{\mathbf{c}} = \mathbf{w} - \mathbf{w}' \in \mathsf{span}_{\mathbb{Z}}(W)$ which is a contradiction. Therefore $|\mathsf{span}_{\mathbb{Z}}(W \cup \{\overline{\mathbf{c}}\})| \geq |A| + |B| = 2 \cdot |\mathsf{span}_{\mathbb{Z}}(W)|$ $\square$

$\square$

Soundness: VBB Simulatability. To derive the soundness we apply Lemma 2.2 as follows. O will be the obfuscator in the ideal model and $\widehat{O}$ will be our obfuscator in the plain model where $z' = Q_B, W, T, R$ is the extra information output by $\widehat{O}$. The algorithm U will be a similar algorithm to $\widehat{O}$ but only during its learning phase 1 and 2 starting from an already obfuscated B. However, U will continue generating z' using the actual oracle $\mathcal{I}[G \mapsto S]$ instead of inventing the answers through lazy evaluation. Since the emulation of the oracle during the learning phases, and that all of Q_B, W, T, R could be obtained by only having B (and no secret information about the obfuscation phase, such as Q are not needed) the algorithm U also has the properties needed for Lemma 2.2.

4 Impossibility of VBB in Degree-$O(1)$ Graded Encoding Model

We prove the following theorem generalizing a similar result by Pass and shelat [Pas15] who proved this for any finite field; here we prove the theorem for any finite ring.

Theorem 4.1. *Let R be any ring of size at most $2^{\mathrm{poly}(n)}$. Let O be any ϵ-approximate VBB obfuscator for general circuits in the ideal degree-d graded encoding model $\mathcal{M}_R^d$ for $d = O(1)$, where the initialization phase of $\mathcal{M}_R^d$ happens during the obfuscation phase. Then for any $\delta = 1/\mathrm{poly}(n)$ there exists an $(\epsilon + \delta)$-approximate obfuscator $\widehat{O}$ for poly-size circuits in the plain model.*

As in previous sections, the following corollary is obtained from Theorem 4.1 by taking $\delta = \epsilon/2$.

Corollary 4.2. *If TDPs exist, then there exists no $(1/2 - \epsilon)$-approximate VBB obfuscator O for general circuits in the ideal degree-d graded encoding model $\mathcal{M}_R^d$ for* any *finite ring R of at most exponential size $|R| \leq 2^{\mathrm{poly}(n)}$ and any constant degree d, assuming the initialization phase of $\mathcal{M}_R^d$ happens during the obfuscation phase.*

They state their theorem in a more general model where a sequence of fields of growing size are accessed. For simplicity, we state a simplified variant for simplicity of presentation where only one ring is accessed but we let the size of ring R to depend on the security parameter n. Our proof follows the footsteps of [Pas15] but will deviate from their approach when $R \neq \mathbb{Z}_p$ by using some of the ideas employed in Section 3.

Proof Sketch. Here we only sketch the proof assuming the reader is familiar with the proof of Theorem 3.1 from previous section. The high level structure of the proof remains the same.

Construction. The new obfuscator $\widehat{O}$ will have the following phases:

- *Emulating obfuscation.* $\widehat{O}$ emulates the randomized obfuscation $O^{\mathcal{M}_R^d}(C)$ to get circuit B.
- *Learning heavy subspace of space of zero vectors*: The learning phase here will be rather simpler than those of Section 3.1 and will be just one phase. Here we repeat the learning phase m times where m is chosen at random from $\ell = [\lceil k \cdot \log(|G|)/\delta \rceil]$. The variables W and T will be the same as in Section 3.1 with the difference that W will consist of the vector of coefficients for all polynomials whose zero test answer is true.
- The returned obfuscated code will be $\widehat{B} = (B, W, T, r)$ where r is again the randomness needed to run the obfuscated code.
- *Executing $\widehat{B}$.* To execute $\widehat{B}$ on input x, we answer zero test queries as follows. For any query vector (of coefficients) $\mathbf{a}$ we test whether $\mathbf{a} \in \mathsf{span}_{\mathbb{Z}}(W)$.[11] If $\mathbf{a} \in \mathsf{span}_{\mathbb{Z}}(W)$ then return true, otherwise return false.

[11]Note that we do *not* solve a system of equations in R and rather search only integer solutions to $\mathbf{x}W = \mathbf{a}$ as we did in Section 3.1.

Completeness and Soundness.

- The completeness follows from the same argument given for the soundness of Construction 3.1. Namely, the execution of $\widehat{B}$ is identical to the execution of the $m+1$'s learning phase (as if it exists) up to a point where we return a wrong false answer to an answer that is indeed a zero polynomial. (Note that the converse never happens). However, when such event is about to happen, the size of $\mathsf{span}_{\mathbb{Z}}(W)$ will double. Since the size of $\mathsf{span}_{\mathbb{Z}}(W)$ is at most $|R|^k$, if we choose m at random from $[\ell]$ the probability of the bad event (of returning a wrong false in $m+1$'st execution) is at most $k \log |R|/\ell = \delta$.

- The soundness follows from Lemma 2.2 similarly to the way we proved the soundness of the construction of Section 3.1.

□

Remark 4.3. Note that our proof did not assume any property for the multiplication (even associativity!) other than being distributive. We indeed need to conclude that the summation of the vectors of the coefficients of two zero polynomials is also the vector of a zero polynomial, which is impled by distributivity.

5 Impossibility of VBB in the Random TDP Oracle Model

In this section we formally prove Theorem 1.3 showing that any obfuscator O with access to a random trapdoor permutation oracle T can be transformed into a new obfuscator $\widehat{O}$ in the plain model (no access to an ideal oracle) with some loss in correctness.

Theorem 5.1 (Theorem 1.3 formalized)**.** *Let O be an ϵ-approximate obfuscator for poly-size circuits in the random TDP oracle model. Then, for any $\delta = 1/\operatorname{poly}(n)$, there exists an $(\epsilon+\delta)$-approximate obfuscator $\widehat{O}$ in the plain model for poly-size circuits.*

Since [BP13] showed that assuming TDPs, no $(1/2 - 1/\operatorname{poly})$-approximate VBB obfuscator does not exist for general circuits, we obtain the following corollary by taking $\delta = \epsilon/2$.

Corollary 5.2. *If TDPs exist, then there exists no $(1/2-\epsilon)$-approximate VBB obfuscator O for general circuits in the ideal TDP model for any $\epsilon = 1/\operatorname{poly}(n)$.*

In the rest of this section, we prove Theorem 5.1. We will first describe the algorithm of the obfuscator in the plain model, and then will analyze its completeness and soundness.

5.1 The Construction

We first describe how the new obfuscator $\widehat{O}$ generates some data $\widehat{B}$, and then we will show how to use $\widehat{B}$ to run the new obfuscated circuit in the plain model. We also let $l_O = \operatorname{poly}(n)$ and l_B be the number of queries asked by the obfuscator O and the obfuscated code B, respectively. To make our discussions on the analysis of TDP queries more robust, we provide the following definition.

Definition 5.3 (TDP query tuple)**.** Given a random l-level HTDP oracle $T_n^l = (\{G_i\}_{i=1}^{l+1}, \{H_i^{-1}\}_{i=0}^{l+1})$, an i-level *TDP query tuple* consists of three (possibly) related query-answer pairs $(V_{H_{i-1}^{-1}}, V_{H_i^{-1}}, V_{G_i})$ where, for any fixed $\mathrm{ID}_{i-2} = (id_0, ..., id_{i-2})$:

- $V_{H_{i-1}^{-1}} = (td_{i-1}, id_{i-1})$ represents a query to $H_{i-1}^{-1}[\mathrm{ID}_{i-3}, id_{i-2}]$ on input td_{i-1} and its corresponding answer id_{i-1}
- $V_{H_i^{-1}} = ((id_{i-1}, td_i), id_i)$ represents a query to $H_i^{-1}[\mathrm{ID}_{i-2}, id_{i-1}]$ on input td_i and its corresponding answer id_i
- $V_{G_i} = ((td_{i-1}, id_i), td'_i)$ represents a query to $G_i[\mathrm{ID}_{i-2}, td_{i-1}]$ on input id_i and its corresponding answer td'_i

We say that an i-level TDP query tuple is *consistent* if $td_i = td'_i$.

Remark 5.4. Note that, for the special case of 1-level HTDP (i.e. TDP) which is of particular interest for the main part of the proof, we only have TDP query tuples of the form $(V_{H_0^{-1}}, V_{H_1^{-1}}, V_{G_1}) = (V_G, V_F, V_F^{-1})$. Thus, $V_G = (sk, pk)$ represents a query to G on $sk = td_0$ and the answer $pk = id_0$, $V_F = ((pk, x), y)$ represents a query to F_{pk} on $x = td_1$ and the answer $y = id_1$, and $V_{F^{-1}} = ((sk, y), x')$ represents a query to F_{sk}^{-1} on y and the answer x', which should be x if the tuple is consistent.

5.1.1 How to Obfuscate

The new obfuscator $\widehat{O}$ in plain model. Given an ϵ-approximate obfuscator O in the random TDP model, we construct a plain-model obfuscator $\widehat{O}$ such that, given a circuit $C \in \{0,1\}^n$, works as follows:

1. *Emulation phase*: Emulate $O^T(C)$. Let Q_O represent the set of queries asked by O^T and their corresponding answers. We initialize $Q_O = \varnothing$. For every query q asked by $O^T(C)$:
   - If q is of the form $F[pk](x)$ for any pk, x then answer with uniformly random y conditioned on any answers to previous queries for $F[pk](.)$.
   - If q is of the form $G(sk)$ for any sk then answer with uniformly random pk conditioned on any answers to previous queries for G.
   - If q is of the form $F^{-1}[sk](y)$ for any sk, y:
     - If query $G(sk)$ is not in Q_O then randomly assign some pk to $G(sk)$ and add $(G(sk), pk)$ to Q_O.
     - If there exists a query-answer pair $(F[pk](x), y)$ in Q_O for some x then set the answer of q to be x. Otherwise, randomly assign some x to q. Add (q, x) to Q_O.
     - If there does not exist a query $F[pk](x)$ then add $(F[pk](x), y)$ to Q_O.
2. *Canonicalize B*: Let the obfuscated circuit B be the output of $O(C)$. Modify B so that, before asking any query of the form $F^{-1}[sk](y)$, it would first ask $G(sk)$ to get some answer pk followed by $F^{-1}[sk](y)$ to get some answer x then finally asks $F[pk](x)$.
3. *Learning phase*: Set $Q_B = \varnothing$. Let the number of iterations to run the learning phase be $m = 2l_B l_O/\delta$ where $l_B \leq |B|$ represents the number of queries asked by B and $l_O \leq |O|$ represents the number of queries asked by O. For $i = [1, m]$:
   - Choose $x_i \xleftarrow{\$} D_{|C|}$

- Run $B(x_i)$. For every query q asked by $B(x_i)$:
    - If $(q, a) \in Q_O \cup Q_C$ for some answer a, answer consistently with a
    - Otherwise, answer q uniformly at random and conditioned on the answers of previous related queries in $Q_O \cup Q_B$
    - Let a be the answer to query q. If $(q, a) \notin Q_B$, add the query-answer pair (q, a) to Q_B

4. The output of the obfuscation algorithm will be $\widehat{B} = (B, Q_B, R)$ where $R = \{r_1, ..., r_{|B|}\}$ is a set of (unused) oracle answers that are generated uniformly at random.

5.1.2 How to Execute

To execute $\widehat{B}$ on an input x using (B, Q_B, R) we simply emulate $B(x)$. For every query q asked by $B(x)$, if $(q, a) \in Q_C$ for some a then return a. Otherwise, answer randomly with one of the answers in R.

5.2 Completeness and Soundness

Completeness: Approximate functionality. Define T to be an ideal TDP oracle. Fix any circuit C of size n, by the approximate functionality of O, we have that:

$$\Pr_x[O^T(C)(x) \neq C(x)] = \Pr_x[B^T(x) \neq C(x)] \leq \epsilon(n)$$

Since the plain-model obfuscated circuit $\widehat{B}$ emulates $B(x)$ on a given input x, we can think of this execution as $B^{\widehat{T}}(x)$ where $\widehat{T}$ is the TDP oracle simulated by $\widehat{B}$ using Q_B and R as the oracle answers (without knowing Q_O, which is part of oracle T). In order to determine the probability of correctness for the plain-model obfuscator $\widehat{O}$, we will identify and isolate the events that differentiate between the executions $B^T(x)$ and $B^{\widehat{T}}(x)$.

- **Bad event E_1:** For any input x, let $E_1(x)$ be the event that $B^{\widehat{T}}(x)$ asks a query q such that $(q, a) \in Q_O \backslash Q_B$ for some answer a (i.e. it asks an unlearned Q_O query).

- **Bad event E_2:** For any input x, let $E_2(x)$ be the event that $B^{\widehat{T}}(x)$ asks a query q such that $(q, a) \notin Q_O \cup Q_B$ for some answer a (i.e. it asks a new query) that forms a TDP query tuple $(V_G, V_F, V_{F^{-1}})$ with a query-answer pair from Q_O. Since q's answer is not in Q_B, it will be answered randomly using R (when its answer is actually related to another query's answer in Q_O) and thus, the resulting TDP query tuple would be inconsistent.

We argue that E_1 and E_2 are the only events that might hurt the correctness of $\widehat{O}$ since the only other cases behave consistently with Q_O: if $(q, a) \in Q_B$ for some answer a then we already know the answer of the query, and if q is a new query that, when answered randomly, does not form an inconsistent TDP query tuple with some query-answer tuples in Q_O then we can do so without affecting the correctness. Thus, assuming that $E_1(x)$ and $E_2(x)$ does not happen, the output distributions of $B^T(x)$ and $B^{\widehat{T}}(x)$ are identical. More formally, the probability of correctness for

$\widehat{O}$ is:

$$\begin{aligned}\Pr_x[B^{\widehat{T}}(x) \neq C(x)] &= \Pr_x[B^{\widehat{T}}(x) \neq C(x) \wedge \neg(E_1(x) \vee E_2(x))] + \Pr_x[B^{\widehat{T}}(x) \neq C(x) \wedge (E_1(x) \vee E_2(x))] \\ &\leq \Pr_x[B^{\widehat{T}}(x) \neq C(x) \wedge \neg(E_1(x) \vee E_2(x))] + \Pr_x[E_1(x) \vee E_2(x)] \\ &= \Pr_x[B^{\widehat{T}}(x) \neq C(x) \wedge \neg(E_1(x) \vee E_2(x))] + \Pr_x[E_1(x)] + \Pr_x[E_2(x) \wedge \neg E_1(x)]\end{aligned}$$

As previously mentioned, we know that

$$\Pr_x[B^{\widehat{T}}(x) \neq C(x) \wedge \neg(E_1(x) \vee E_2(x))] = \Pr_x[B^{T}(x) \neq C(x) \wedge \neg(E_1(x) \vee E_2(x))] \leq \epsilon$$

Therefore, we are left to find $\Pr[E_1(x)]$ and $\Pr[E_2(x) \wedge \neg E_1(x)]$. The probability of event E_1 was already given in [CKP15], but for the sake of completeness we show our version of the analysis here.

Claim 5.5.

$$\Pr_x[E_1(x)] \leq \delta/2$$

Proof of Claim 5.5. Let $(q_1, ..., q_{l_B})$ be the sequence of queries asked by $B^{\widehat{T}}(x)$ where $l_B \leq |B|$, and let $q_{i,j}$ be the j^{th} query that is asked by $B^T(x_i)$ during the i^{th} iteration of the learning phase. We define $E_1^j(x)$ to be the event that the j^{th} query of $B(x)$ is in Q_O but not in Q_B. We also define $p_{q,j}$ to be the probability that $q_j = q$ for any query q and $j \in [l_B]$. We can then write the probability of E_1 as follows:

$$\begin{aligned}\Pr_x[E_1(x)] &\leq \Pr_x[E_1^1(x) \vee ... \vee E_1^{l_B}(x)] \\ &= \sum_{j=1}^{l_B} \Pr_x[\neg E_1^1(x) \wedge ... \wedge \neg E_1^{j-1}(x) \wedge E_1^j(x)] \\ &\leq \sum_{j=1}^{l_B} \sum_{q \in Q_O} \Pr_x[q_j = q \wedge (q_{1,j} \neq q \wedge ... \wedge q_{m,j} \neq q)] \\ &= \sum_{j=1}^{l_B} \sum_{q \in Q_O} p_{q,j}(1 - p_{q,j})^m \\ &\leq \sum_{j=1}^{l_B} \sum_{q \in Q_O} \frac{1}{m} \\ &\leq \sum_{j=1}^{l_B} \frac{l_O}{m} = \frac{l_B l_O}{m}\end{aligned}$$

Thus, given that $m = 2l_B l_O/\delta$, we get $\Pr[E_1(x)] \leq \delta/2$. □

Claim 5.6.

$$\Pr_x[E_2(x) \wedge \neg E_1(x)] \leq l_B l_O/2^n$$

Proof of Claim 5.6. This entails finding all possible query cases where a new evaluation of $B(x)$ *does not* hit a query in $Q_O \setminus Q_B$ and yet still asks a query that is inconsistent with one or more queries in $Q_O \setminus Q_B$.

Due to the canonicalization procedure, Q_O can only have four possible types of TDP query tuples: $(V_G, \cdot, \cdot), (\cdot, V_F, \cdot), (V_G, V_F, \cdot)$, and $(V_G, V_F, V_{F^{-1}})$. This is because whenever O asks a query for F^{-1}, $\widehat{O}$ will be forced to learn all the queries that are related to this query and add them all to Q_O. Assuming that E_1 does not happen, we examine which of the three possible queries (G, F, and F^{-1}) initiated by a new evaluation of B will possibly yield an inconsistent TDP query tuple when compared against the aforementioned four types of TDP tuples in Q_O. Let $q \notin Q_B$ be a new query that B asks when it is emulated by $\widehat{B}$ on some new input [12]:

- If q is of the form $F[pk](x)$ for any pk, x then the only possible TDP query in $Q_O \setminus Q_B$ that it might relate to is $(V_G, \cdot, \cdot)$ for some $V_G = (sk, pk)$ since E_1 does not occur and therefore $V_F = ((pk, x), y)$ cannot be in $Q_O \setminus Q_B$. As a result, $\widehat{B}$ can answer q randomly (using R) without generating an inconsistent TDP tuple.

- If q is of the form $G(sk)$ for any sk then the only possible TDP query in $Q_O \setminus Q_B$ that it might relate to is $(\cdot, V_F, \cdot)$ for some $V_F = ((pk, x), y)$ since E_1 does not occur and therefore $V_G = (sk, pk)$ cannot be in $Q_O \setminus Q_B$. As a result, $\widehat{B}$ can answer q randomly (using R) without generating an inconsistent TDP tuple.

- If q is of the form $F^{-1}[sk](y)$ for any sk, y then, as per our modification of B, it will first be preceded by a query $G(sk)$, which will be randomly answered with pk using R or, if this query was previously invoked, using the already stored answer. Once this is done, we answer query $F^{-1}[sk](y)$ using some random answer x and ask query $F[pk](x)$, which will be set by $\widehat{B}$ to be y. Since E_1 does not occur, $Q_O \setminus Q_B$ cannot contain a TDP query tuple with $V_G = (sk, pk)$, $V_{F^{-1}} = ((sk, y), x)$, or $V_F = ((pk, x), y)$. However, it might possess a TDP query tuple of the form $(\cdot, V'_F, \cdot)$ where $V'_F = ((pk, x'), y)$. In that case, when we merge this query tuple with the obfuscated circuit's $V_G = (sk, pk)$ and $V_{F^{-1}} = ((sk, y), x)$, we get an inconsistent TDP query tuple $(V_G, V'_F, V_{F^{-1}})$.

We can therefore see that the only possibility that can lead to an inconsistency is the third case above. Since $G(sk)$ is answered uniformly at random with some pk using one of the elements in R, the probability of hitting $y = F[pk](x')$ when asking $F^{-1}[sk](y)$ and inducing an inconsistent tuple is at most $l_O/2^n$ since the value of the chosen pk coincides with pk' for at most l_O queries (made by O) of the form $F[pk'](.)$. Finally, by a union bound, the probability that any query asked by B results in an inconsistent query tuple is at most $l_B l_O/2^n$, which is a negligible function in n since both l_B and l_O are polynomials. □

Soundness: VBB Simulatability. To show that the security property is satisfied, it suffices to provide a PPT algorithm U^T in the ideal TDP model that takes as input $O^T(C)$ for some circuit C and outputs a distribution that is statistically close to the output distribution of $\widehat{O}$. If that is the case, we can invoke Lemma 2.2 and conclude that $\widehat{O}$ is also VBB-secure.

[12]We assume without loss of generality that the query asked by B is not in Q_B since otherwise we can answer these queries consistently using the answers in Q_B.

The description of U is precisely the same as Steps 2-4 of the procedure detailed in Section 5.1 except that queries made by $B = O^T(C)$ are answered using oracle T instead of being randomly simulated. If we let (B, Q_B, R) be the output of $U^T(O^T(C))$ then we can easily see that it is identically distributed to the output distribution of $\widehat{O}$ since, in both cases, Q_B has query-answers with consistent and random TDP query tuples. They differ only by how these query answers are generated (U^T answers these queries using T, while $\widehat{O}$ simulates them using lazy evaluation with respect to some oracle $\widehat{T}$ that is distributed the same as T).

5.3 Extension to hierarchical random TDP

The extension of the impossibility result to random HTDP is straightforward, so we will outline the main differences between the TDP case and describe how to resolve the issues that are related to this oracle.

First, we still perform the normalisation procedure on $\widehat{O}$ and B where the query behaviour of these algorithms are modified such that for any query q of the form $G_i[\mathrm{ID}_{i-2}, td_{i-1}](id_i)$, we first ask $H_{i-1}^{-1}[\mathrm{ID}_{i-3}, id_{i-2}](td_{i-1})$ to get id_{i-1}. This is allows us to discover whether we have a query $H_i^{-1}[\mathrm{ID}_{i-2}, id_{i-1}](td_i)$ whose answer is id_i, in which case we can answer q with td_i. This procedure ensures that all query tuples that contain G_i queries are consistent.

We now turn to verifying whether the proof of approximate functionality for TDP holds in this case as well and, in particular, focus on the event of $(E_2 \wedge \neg E_1)$. The main issue that we have to consider, which is unique to the HTDP case, is the possibility that different consistent TDP query tuples can be related to each other, and an overlap between these queries may cause an inconsistency in one of the tuples. Specifically, an i-level TDP query tuple of the form $(V_{H_{i-1}^{-1}}, \cdot, \cdot)$ might overlap with an $(i-1)$-level TDP query tuple $(\cdot, \cdot, V_{G_{i-1}})$ from Q_O, where the answer of $V_{H_{i-1}^{-1}}$ is inconsistent with that of $V_{G_{i-1}}$. However, our normalisation procedure prevents precisely this issue as any TDP query tuple that contains $V_{G_{i-1}}$ must also have $V_{H_{i-1}^{-1}}$, which means that the queries should not overlap otherwise event E_1 happens leading to a contradiction to our initial assumption.

Acknowledgement. We thank Victor Shoup and Hendrik W. Lenstra for pointing us out to the literature on solving linear equations over the ring $\mathbb{Z}_n$.

References

[BGI+01] Boaz Barak, Oded Goldreich, Rusell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im) possibility of obfuscating programs. In *Advances in cryptologyCRYPTO 2001*, pages 1–18. Springer, 2001. 1

[BGK+13] Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Protecting obfuscation against algebraic attacks. *IACR Cryptology ePrint Archive*, 2013:631, 2013. 7

[BGK+14] Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Protecting obfuscation against algebraic attacks. In *Advances in Cryptology–EUROCRYPT 2014*, pages 221–238. Springer, 2014. 1, 2, 9

[BP13] Nir Bitansky and Omer Paneth. On the impossibility of approximate obfuscation and applications to resettable cryptography. In *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, STOC '13, pages 241–250, New York, NY, USA, 2013. ACM. 1, 3, 11, 18

[BR14] Zvika Brakerski and Guy N Rothblum. Virtual black-box obfuscation for all circuits via generic graded encoding. In *Theory of Cryptography*, pages 1–25. Springer, 2014. 1, 2, 9

[Can97] Ran Canetti. Towards realizing random oracles: Hash functions that hide all partial information. In *Advances in CryptologyCRYPTO'97*, pages 455–469. Springer, 1997. 1

[CGH04] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004. 1

[CKP15] Ran Canetti, Yael Tauman Kalai, and Omer Paneth. On obfuscation with random oracles. Cryptology ePrint Archive, Report 2015/048, 2015. `http://eprint.iacr.org/`. 1, 2, 3, 4, 6, 7, 21

[GGH13a] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In *Eurocrypt*, volume 7881, pages 1–17. Springer, 2013. 1

[GGH$^+$13b] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Anant Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, pages 40–49. IEEE, 2013. 1

[HL02] Jeremy Horwitz and Ben Lynn. Toward hierarchical identity-based encryption. In LarsR. Knudsen, editor, *Advances in Cryptology EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 466–481. Springer Berlin Heidelberg, 2002. 2

[IR89] Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing (STOC)*, pages 44–61. ACM Press, 1989. 2

[LPS04] Benjamin Lynn, Manoj Prabhakaran, and Amit Sahai. Positive results and techniques for obfuscation. In *Advances in Cryptology-EUROCRYPT 2004*, pages 20–39. Springer, 2004. 1

[McC90] Kevin S. McCurley. The discrete logarithm problem. In *Proc. of the AMS Symposia in Applied Mathematics: Computational Number Theory and Cryptography*, pages 49–74. American Mathematical Society, 1990. 4, 5, 25

[Pas15] Rafael Pass and abhi shelat. Impossibility of vbb obfuscation with ideal constant-degree graded encodings. Cryptology ePrint Archive, Report 2015/383, 2015. `http://eprint.iacr.org/`. 1, 2, 3, 4, 5, 6, 9, 11, 17

[RTV04] Omer Reingold, Luca Trevisan, and Salil P. Vadhan. Notions of reducibility between cryptographic primitives. In *Theory of Cryptography, First Theory of Cryptography Conference, TCC 2004*, volume 2951 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2004. 2

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *Advances in Cryptology EUROCRYPT 97*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266. Springer Berlin Heidelberg, 1997. 2, 8, 9

[Wee05] Hoeteck Wee. On obfuscating point functions. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 523–532. ACM, 2005. 1

A Solving Linear Equations over Abelian Groups

Here we describe a polynomial time algorithm for iLEAG as defined in Definition 3.8.

Theorem A.1. *iLEAG can be solved in polynomial time.*

Cyclic vs General Abelian. Note that if $G = \mathbb{Z}_p$ for a prime p iLEAG can be solved efficiently using Guassian elimination. If the abelian group G is cyclic $\mathbb{Z}_q$, it does not matter if we define the variables $\mathbf{x}$ in $\mathbb{Z}^n$ or in $\mathbb{Z}_q^n$. In this case, if q is prime, we can use the Gaussian elimination method to find $\mathbf{x}$. Interestingly, even if q is not a prime, there is still a way to solve systems of linear equations over $\mathbb{Z}_q$ using a generalization of Gaussian elimination [McC90] which is the main tool we use to prove Theorem A.1.

Theorem A.2 ([McC90] Section 5.1)**.** *Systems of linear equations over $\mathbb{Z}_q$ for integer q can be solved in polynomial time. I.e., iLEAG is solvable in polynomial for cyclic abelian G.*

In the rest of this section we prove Theorem A.1. We will do so by reducing iLEAG to $G = \mathbb{Z}_q$ (where q happens to be a prime power) and applying Theorem A.2.

Proof of Theorem A.1 using Theorem A.2. We will change LAEG in two steps by defining simplified versions of it and reducing it to these simplified forms.

Definition A.3 (Linear Equations over Arbitrary Prime Powers (LEAP))**.** Let $\{\mathbf{a}^j \mid j \in [m], \mathbf{a}^j \in \mathbb{Z}^n\}$ be a set of m vertical vectors in $\mathbb{Z}^n$ and suppose we are given $p_i^{\alpha_i}$ in which p_i is a prime (but it could be that $p_i = p_j$). We want to find an integer vector $x \in \mathbb{Z}^n$ such that for all $j \in [m]$ the inner product $\langle \mathbf{x}, \mathbf{a}^j \rangle$ is zero mod $p_j^{\alpha_j}$.

Claim A.4. *Solving iLEAG can be reduced to solving LEAP in polynomial time.*

Proof. Let $A = [\mathbf{a}^1, \dots, \mathbf{a}^k], \mathbf{b} = (b_1 \dots, b_k)$ be a given instance of iLEAG over G. Let G be isomorphic to $\mathbb{Z}_{p_1^{\alpha_1}} \times \cdots \times \mathbb{Z}_{p_d^{\alpha_d}}$. Then we can think of every column $\mathbf{a}^i$ in A as an $n \times d$ matrix $C_i = [\mathbf{c}_i^1, \dots, \mathbf{c}_i^d]$ for which $\langle \mathbf{x}, \mathbf{a}^i \rangle = b_i$ is equivalent to having $\langle \mathbf{x}, \mathbf{c}_i^j \rangle = 0 \mod p_j^{\alpha_j}$ for every $j \in [d]$. And so $\mathbf{c}A = b$ will be equivalent to $\langle \mathbf{x}, \mathbf{c}_i^j \rangle = 0 \mod p_j^{\alpha_j}$ for all $i \in [k]$ and $j \in [d]$. The latter is indeed an iLEDM instance (with $k \cdot d$ linear constraints over a total of d different moduli). □

Definition A.5 (Linear Equations over Identical Prime Powers (LEIP))**.** This problem is a special case of LEAP in which all the moduli are over the same prime, but perhaps with different powers. Namely, if $p_i^{\alpha_i}$ and $p_j^{\alpha_j}$ is the modulus for the i inner product, we have $p_i^{\alpha_i} = p_j^{\alpha_j}$ for all $i, j \in [m]$.

Claim A.6. *Solving LEAP could be reduced to solving LEIP in polynomial time.*

Proof. Let m be the number of linear constraints and $p_i^{\alpha_i}$ be the modulus for that restriction. Partition $[m]$ into sets $S_1, \ldots, S_\ell$ such that i, j are in the same partition S_k if and only if $p_i = p_j$. This partitioning defines ℓ subproblems from the original LEAP instance that we simply denote by $S_1, \ldots, S_\ell$. It is easy to see that any solution $\mathbf{x}$ to the original LEAP instance also works for all of its sub problems. We prove the converse using the Chinese reminder theorem.

Let $q_j = \max_{i \in S_j} p_i^{\alpha_i}$ be the largest modulus whose index is in S_j. For every $i \in S_j$ let $r = q_j / p_i^{\alpha_i}$ and then multiply $\mathbf{a}^i$, b_i, as well as the modulus restriction for them (i.e., $p_i^{\alpha_i}$) by r. Note that this will not change the space of solutions, but will make all the modulus restrictions in S_j equal to q_j.

Now let $\mathbf{x}_j$ be the integer solution to LEIP instance S_j for $j \in [\ell]$. Using the (algorithmic proof of the) Chinese reminder theorem we can use $\mathbf{x}_j$'s to find a single integer vector $\mathbf{x}$ such that $\mathbf{x} = \mathbf{x}_j \mod q_j$ for all $j \in [\ell]$ which means $\mathbf{x}$ is an integer solution to the original LEAP instance. □

Finally note that LEIP is a special case of solving linear equations over $\mathbb{Z}_q$ for general integer q which can be solved in polynomial time according to Theorem A.2.

□

Finally, we note that there is also a polynomial time algorithm for the arguably "more natural" problem G-LEAG (see Remark 3.9) in which we seek solutions for $\mathbf{x}A = \mathbf{b}$ *inside* ring G where the multiplication in G is defined according to Definition 3.5. Note that since the multiplication of Definition 3.5 generalizes the notion of ring $\mathbb{Z}_n$, the following theorem is a also a generalization of solving system of linear equations over $\mathbb{Z}_n$, as was the case for iLEAG, but here the generalization is in a different direction.

Theorem A.7. *G-LEAG can be solved in polynomial time for finite abelian groups.*

Proof. As opposed to the case of iLEAG we show how to reduce G-LEAG directly to LEIP. Suppose we are given an $n \times k$ matrix A and a $1 \times k$ vector $\mathbf{b}$ with all elements in G, and we want to find $\mathbf{x}$ such that $\mathbf{x}A = \mathbf{b}$. Let $G = \mathbb{Z}_{p_1^{\alpha_1}} \times \cdots \times \mathbb{Z}_{p_d^{\alpha_d}}$. Therefore all elements of $A, \mathbf{b}$ and even $\mathbf{x}$ could be thought as $1 \times d$ vectors were the addition and multiplication in the i'th coordinate is done modulo $p_i^{\alpha_i}$. Now let $(A^i, \mathbf{b}^i)$ be the LEIP instance that could be derived from $\mathbf{x}A = \mathbf{b}$ and for each element we substitute the corresponding i'th coordinate in its vector representation of G. It is easy to see that solving the G-LEAG instance is equivalent to solving *all* of the d LEIP subproblems $(A^i, \mathbf{b}^i)$ for $i \in [d]$ and simply using the solutions $\mathbf{x}^i$ (whose components are modulo $p_i^{\alpha_i}$) to scale them up into a solution $\mathbf{x}$ in G. □