Annihilation Attacks for Multilinear Maps: Cryptanalysis of Indistinguishability Obfuscation over GGH13

Eric Miles
UCLA
enmiles@cs.ucla.edu

Amit Sahai*
UCLA
sahai@cs.ucla.edu

Mark Zhandry
MIT
mzhandry@gmail.com

Abstract

In this work, we present a new class of polynomial-time attacks on the original multilinear maps of Garg, Gentry, and Halevi (2013). Previous polynomial-time attacks on GGH13 were "zeroizing" attacks that generally required the availability of low-level encodings of zero. Most significantly, such zeroizing attacks were not applicable to candidate indistinguishability obfuscation (iO) schemes. iO has been the subject of intense study.

To address this gap, we introduce *annihilation attacks*, which attack multilinear maps using non-linear polynomials. Annihilation attacks can work in situations where there are no low-level encodings of zero. Using annihilation attacks, we give the first polynomial-time cryptanalysis of candidate iO schemes over GGH13. More specifically, we exhibit two simple programs that are functionally equivalent, and show how to efficiently distinguish between the obfuscations of these two programs.

Given the enormous applicability of iO, it is important to devise iO schemes that can avoid attack. We discuss some initial directions for safeguarding against annihilating attacks.

*Research supported in part from a DARPA/ARL SAFEWARE award, NSF Frontier Award 1413955, NSF grants 1228984, 1136174, 1118096, and 1065276, a Xerox Faculty Research Award, a Google Faculty Research Award, an equipment grant from Intel, and an Okawa Foundation Research Grant. This material is based upon work supported by the Defense Advanced Research Projects Agency through the ARL under Contract W911NF-15-C-0205. The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense, the National Science Foundation, or the U.S. Government.

1 Introduction

In this work, we present a new class of polynomial-time attacks on the original multilinear maps of Garg, Gentry, and Halevi [GGH13a]. Previous attacks on GGH13 were not applicable to many important applications of multilinear maps, most notably candidate indistinguishability obfuscation (iO) schemes over GGH13 [GGH+13b, BR14, BGK+14, AGIS14, MSW14, BMSZ15]. Indeed, previous attacks on GGH13 can be classified into two categories:

- Works presenting polynomial-time attacks that either explicitly required the availability of low-level encodings of zero [GGH13a, HJ15], or required a differently represented low-level encoding of zero, in the form of an encoded matrix with a zero eigenvalue [CGH+15]. As a result, such "zeroizing" attacks do not apply to any iO candidates.
- Works that yield subexponential or quantum attacks [CDPR15, CJL16]. This includes the work of [CJL16] that was announced concurrently with the initial publication of our work. We note that the attacks of [CJL16] on GGH13 mmaps, for example, require exponential running time if $n = \lambda \log^2 q$, and these attacks do not apply at all if n is not a power of 2.

iO has been the subject of intense study. Thus, understanding the security of candidate iO schemes is of high importance. To do so, we need to develop new polynomial-time attacks that do not require, explicitly or implicitly, low-level encodings of zero.

Annihilation Attacks. To address this gap, we introduce *annihilation attacks*, which attack multilinear maps in a new way, using non-linear polynomials. Annihilation attacks can work in situations where there are no low-level encodings of zero. Using annihilation attacks, we give the first polynomial-time cryptanalysis of several candidate iO schemes over GGH13 from the literature. More specifically, we exhibit two simple programs that are functionally equivalent, and show how to efficiently distinguish between the obfuscations of these two programs. We also show how to extend our attacks to more complex candidate obfuscation schemes over GGH13, namely ones that incorporate the "dual-input" approach of [BGK+14]. (Note that, even without the dual-input structure, [BGK+14, AGIS14, MSW14, BMSZ15] were candidates for achieving iO security when implemented with [GGH13a].) Additionally, we give the first polynomial-time cryptanalysis of the candidate order revealing encryption scheme due to Boneh et al. [BLR+15] when instantiated over GGH3.

We now give an overview of our attack. The overview will introduce the main conceptual ideas and challenges in mounting our attack. After the overview, we will discuss potential defenses that may thwart our attack and generalizations of it.

1.1 Overview of the Attack

We begin with a simplified description of the GGH13 scheme, adapted from text in [CGH+15].

1.1.1 The GGH13 scheme

For GGH13 [GGH13a] with k levels of multilinearity, the plaintext space is a quotient ring $R_g = R/gR$ where R is the ring of integers in a number field and $g \in R$ is a "small element" in that ring. The space of encodings is $R_q = R/qR$ where q is a "big integer". An instance of the scheme relies on two secret elements, the generator g itself and a uniformly random denominator $z \in R_q$. A small plaintext element α is encoded "at level one" as $u = [e/z]_q$ where e is a "small element" in the coset of α, that is $e = \alpha + gr$ for some small $r \in R$.

Addition/subtraction of encodings at the same level is just addition in R_q, and it results in an encoding of the sum at the same level, so long as the numerators do not wrap around modulo q. Similarly multiplication of elements at levels i, i' is a multiplication in R_q, and as long as the numerators do not wrap around modulo q the result is an encoding of the product at level $i + i'$.

The scheme also includes a "zero-test parameter" in order to enable testing for zero at level k. Noting that a level-k encoding of zero is of the form $u = [gr/z^k]_q$, the zero-test parameter is an element of the form $\mathbf{p}_{\mathrm{zt}} = [hz^k/g]_q$ for a "somewhat small element" $h \in R$. This lets us eliminate the z^k in the denominator and the g in the numerator by computing $[\mathbf{p}_{\mathrm{zt}} \cdot u]_q = h \cdot r$, which is much smaller than q because both h, r are small. If u is an encoding of a non-zero α, however, then multiplying by $\mathbf{p}_{\mathrm{zt}}$ leaves a term of $[h\alpha/g]_q$ which is not small. Testing for zero therefore consists of multiplying by the zero-test parameter modulo q and checking if the result is much smaller than q.

Note that above we describe the "symmetric" setting for multilinear maps where there is only one z, and its powers occur in the denominators of encodings. More generally, we will equally well be able to deal with the "asymmetric" setting where there are multiple z_i. However, we omit this generalization here as our attack is agnostic to such choices. Our attack is also agnostic to other basic parameters of the GGH13, including the specific choice of polynomial defining the ring R.

1.1.2 Setting of our attack

Recall that in our setting, we – as the attacker – will not have access to any low-level encodings of zero. Thus, in general, we are given as input a vector $\vec{u}$ of ℓ encodings, corresponding to a vector $\vec{\alpha}$ of ℓ values being encoded, and with respect to a vector $\vec{r}$ of ℓ random small elements. Thus, for each $i \in [\ell]$, there exists some value $j_i < k$ such that

$$u_i = \left[\frac{\alpha_i + gr_i}{z^{j_i}}\right]_q \qquad \alpha_i \neq 0$$

What a distinguishing attack entails. In general, we consider a situation where there are two distributions over vectors: $\vec{\alpha}^{(0)}$ and $\vec{\alpha}^{(1)}$. Rather than thinking of these vectors as directly giving distributions over values, we can think of them as distinct vectors of multivariate polynomials over some underlying random variables. Thus, from this viewpoint, $\vec{\alpha}^{(0)}$ and $\vec{\alpha}^{(1)}$ are just two distinct vectors of polynomials, that are known to us in our role as attacker.

Then a challenger chooses a random bit $b \in \{0, 1\}$, and we set $\vec{\alpha} = \vec{\alpha}^{(b)}$. Then we are given encodings $\vec{u}$ of the values $\vec{\alpha}$ using fresh randomness $\vec{r}$, and our goal in mounting an attack is to determine the challenger's bit b.

Note that to make this question interesting, it should be the case that all efficiently computable methods of computing top-level encodings of zero using encodings of $\vec{\alpha}^{(0)}$ should also yield top-level encodings of zero using encodings of $\vec{\alpha}^{(1)}$. Otherwise, an adversary can distinguish between these encodings simply by zero testing.

1.1.3 Using annihilating polynomials

Our attack first needs to move to the polynomial ring R. In order to do so, the attack will need to build top-level encodings of zero, and then multiply by the zero-testing element $\mathbf{p}_{\mathrm{zt}}$. Because we are in a setting where there are no low-level encodings of zero, top-level encodings of zero can only be created through algebraic manipulations of low-level encodings of nonzero values that lead to cancellation. Indeed, a full characterization of exactly how top-level encodings of zero can be created for candidate iO schemes over GGH13 was recently given by [BMSZ15]. In general, our attack will need to have access to a collection of valid algebraic manipulations that yield top-level encodings of zero, starting with the encodings $\vec{u}$.

Generally, then, a top-level encoding of zero e produced in this way would be stratified into levels corresponding to different powers of g, as follows:

$$e = \frac{g\gamma_1 + g^2\gamma_2 + \cdots g^k\gamma_k}{z^k}$$

and thus

$$f := [e \cdot \mathbf{p}_{\mathrm{zt}}]_q = h \cdot (\gamma_1 + g\gamma_2 + \cdots g^{k-1}\gamma_k)$$

Above, each γ_i is a polynomial in the entries of $\vec{\alpha}$ and $\vec{r}$. As suggested by the stratification above, our main idea is to focus on just one level of the stratification. In particular, let us focus on the first level of the stratification, corresponding to the polynomial γ_1.

A simple illustrative example. Suppose that we had three ways of generating top-level encodings of zero, e, e', and e'', which yield products f, f', and f'' in the ring R. Suppose further that e, e', and e'' contained polynomials $\gamma_1 = xr$; $\gamma_1' = xr^2$; and $\gamma_1'' = x$, where x is a random variable underlying $\vec{\alpha}^{(0)}$. Then we observe that there is an efficiently computable *annihilating polynomial*, $Q(a, b, c) := a^2 - bc$, such that $Q(\gamma_1, \gamma_1', \gamma_1'')$ is the zero polynomial. Further, because Q is homogeneous, $Q(h \cdot \gamma_1, h \cdot \gamma_1', h \cdot \gamma_1'')$ is also the zero polynomial. (We will always ensure that our annihilating polynomials are homogeneous, which essentially comes for free due to the homogeneity of the γ_1 polynomials in the iO setting; see Lemma 5.3.)

Thus, if we compute $Q(f, f', f'')$, we obtain an element in the ring R that is contained in the ideal $\langle hg \rangle$.

However, consider the top-level encodings of zero e, e', and e'' that arise from $\vec{\alpha}^{(1)}$, which is a different vector of polynomials over x than $\vec{\alpha}^{(0)}$. Suppose that in this case, the encodings e, e', and e'' contain polynomials $\gamma_1 = x^3r$; $\gamma_1' = xr$; and $\gamma_1'' = x$. In this scenario, the polynomial Q is no longer annihilating, and instead yields $Q(\gamma_1, \gamma_1', \gamma_1'') = x^6r^2 - x^2r$. Thus, what we have is that if the challenge bit $b = 0$, then $Q(f, f', f'')$ is contained in the ideal $\langle hg \rangle$, but if the challenge bit $b = 1$, then $Q(f, f', f'')$ is not contained in the ideal $\langle hg \rangle$.

Obtaining this distinction in outcomes is the main new idea behind our attack.

1.1.4 Central challenge: How to compute annihilating polynomials?

While it was easy to devise an annihilating polynomial for the polynomials contained in the simple example above, in general annihilating polynomials can be hard to compute. Every set of $n + 1$ or more polynomials over n variables is algebraically dependent and hence must admit an annihilating polynomial. Indeed, therefore, if we do not worry about how to compute annihilating polynomials, our high-level attack idea as described above would apply to *every* published iO scheme that can be instantiated with GGH13 maps that we are aware of, and it would work for every pair of equivalent programs that output zero sufficiently often. This is simply because every published iO candidate can be written as an algebraic expression using only a polynomial number of underlying random variables, whereas the obfuscated program can be evaluated on an exponential number of inputs.

However, unless the polynomial hierarchy collapses (specifically, unless **Co-NP** $\subseteq$ **AM**), there are sets of (cubic) polynomials over n variables for which the annihilating polynomial cannot be represented by any polynomial-size arithmetic circuit [Kay09]. As a result, for our attack idea to be meaningful, we must show that the annihilating polynomials we seek are efficiently representable by arithmetic circuits and that such representations are efficiently computable. In particular, we seek to do this in the context of (quite complex) candidates for indistinguishability obfuscation.

We begin by looking deeper at the structure of the polynomials γ_1 that we need to annihilate. In particular, let's examine what these polynomials look like as a consequence of the stratification by powers of g. We see that by the structure of encodings in GGH13, each polynomial γ_1 will be linear in the entries of $\vec{r}$ and potentially non-linear in the entries of $\vec{\alpha}$. This is already useful, since the $\vec{r}$ variables are totally unstructured and unique to each encoding given out, and therefore present an obstacle to the kind of analysis that will enable us to find an annihilating polynomial.

To attack iO, we will first design two simple branching programs that are functionally equivalent but distinct as branching programs. To this end, we consider two branching programs that both compute the *always zero* functionality. The simplest such program is one where every matrix is simply the identity matrix, and this will certainly compute the constant zero functionality. To design another such program, we observe that the anti-identity matrix

$$B = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

can be useful, because it has the property that $BB = I$. Thus, to make another branching program that computes the always zero functionality, we can create a two-pass branching program, where the (two) matrices corresponding to $x_1 = 0$ are both set to B, and all other matrices are set to I.

With these branching programs in mind, we analyze the γ_1 polynomials that arise. The main method that we use to prune the search space for annihilating polynomials is to find changes of variables that can group variables together in order to minimize the number of active variables. We use a number of methods, including inclusion-exclusion formulas, to do this. By changing variables, we are able to reduce the problem to finding the annihilating polynomial for a set of polynomials over only a *constant* number of variables. When only a constant number of variables are present, exhaustive methods for finding annihilating polynomials are efficient. For further details, refer to Section 5.

Moving to more complex iO candidates. The above discussion covers the main ideas for finding annihilating polynomials, and by generalizing our methods, we show that they extend to more challenging settings. Most notably, we can extend our methods to work for the *dual-input* technique of [BGK$^+$14], which has been used in several follow-up works [AGIS14, MSW14, BMSZ15]. Previously, no cryptanalysis techniques were know to apply to this setting. For further details, see Sections 4 and 5.

1.1.5 An abstract attack model

We first describe our attacks within a new abstract attack model, which is closely related to a model proposed in [CGH$^+$15, App. A]. The new model is roughly the same as existing generic graded encoding models, except that a successful zero test returns an *algebraic element* rather than a bit $b \in \{0,1\}$. These algebraic elements can then be manipulated, say, by evaluating an annihilating polynomial over them. This model captures the fact that, in the GGH13 candidate graded encoding scheme [GGH13a], the zero test actually does return an algebraic element in a polynomial ring that can be manipulated.

We describe our attacks in this abstract model to (1) highlight the main new ideas for our attack, and (2) to demonstrate the robustness of our attack to simple "fixes" for multilinear maps that have been proposed.

Theorem 1.1. *Let $\mathcal{O}$ denote the single-input variant of the iO candidates in [BGK+14, AGIS14, BMSZ15] (over GGH13 [GGH13a] maps). There exist two functionally-equivalent branching programs $\mathbf{A}, \mathbf{A}'$ such that $\mathcal{O}(\mathbf{A})$ and $\mathcal{O}(\mathbf{A}')$ can be efficiently distinguished in the abstract attack model described in Section 2.*

Note that in the single input case, the [BGK+14, AGIS14, BMSZ15] obfuscators over GGH13 [GGH13a] maps were shown to achieve iO security in the standard generic graded encoding model. This theorem shows that such security does not extend to our more refined model.

The attack in Theorem 1.1 works by executing the obfuscated program honestly on several inputs, which produces several zero-tested top-level 0-encodings. Recall that in our model, each successful zero-test returns an algebraic element. We then give an explicit polynomial that annihilates these algebraic elements in the case of one branching program, but fails to annihilate in the other. Thus by evaluating this polynomial on the algebraic elements obtained and testing for zero, it is possible to distinguish the two cases.

1.1.6 Beyond the abstract attack model

Our abstract attack does not immediately yield an attack on actual graded encoding instances. For example, when the graded encoding is instantiated with [GGH13a], the result of an annihilating polynomial is an element in the ideal $\langle hg \rangle$, whereas if the polynomial does not annihilate, then the element is not in this ideal. However, this ideal is not explicitly known, so it is not a priori obvious how to distinguish the two cases.

We observe that by evaluating the annihilating polynomial many times on different sets of values, we get many different vectors in $\langle hg \rangle$. With enough vectors, we (heuristically) can compute a spanning set of vectors for $\langle hg \rangle$. This is the only heuristic portion of our attack analysis, and it is similar in spirit to previous heuristic analysis given in other attacks of multilinear maps (see, e.g., [CGH+15]). With such a spanning set, we can then test to see if another "test" vector is in this ideal or not. This is the foundation for our attack on obfuscation built from the specific [GGH13a] candidate.

Dual-Input Obfuscation and Beyond. Moving on to the dual-input setting, we do not know an explicit annihilating polynomial for the set of algebraic elements returned by our model. However, we are able to show both that such a polynomial must exist, and furthermore that it must be efficiently computable because it has constant size. Thus we demonstrate that there *exists* an efficient distinguishing adversary in the abstract attack model. As before, we can turn this into a heuristic attack on obfuscation built from [GGH13a] graded encodings. We also show that modifying the branching programs to read $d > 2$ bits at each level does not thwart the attack for constant d, because the annihilating polynomial still has constant size (albeit a larger constant).

Theorem 1.2. *Let $\mathcal{O}$ denote the dual-input variant of the iO candidates found in [BGK+14, AGIS14, BMSZ15] (over GGH13 [GGH13a] maps). There exist two functionally-equivalent branching programs $\mathbf{A}, \mathbf{A}'$ such that $\mathcal{O}(\mathbf{A})$ and $\mathcal{O}(\mathbf{A}')$ can be efficiently distinguished in the abstract attack model described in Section 2.*

1.2 Attacking Candidate Order-Revealing Encryption

We show how to apply annihilating polynomials to the candidate order-revealing encryption (ORE) scheme of Boneh et al. [BLR+15]. ORE is a symmetric key encryption scheme where it is possible to learn the order of plaintexts without knowing the secret key, but nothing else is revealed by the ciphertexts. Such a scheme would allow, for example, making range queries on an encrypted database without the secret key. The ORE of [BLR+15] is one of the few implementable applications of multilinear maps. We demonstrate a polynomial such that whether or not the polynomial annihilates depends on more than just the order of the plaintexts. We therefore get an attack in our refined abstract model:

Theorem 1.3. *Let $\mathcal{E}$ denote the ORE scheme of [BLR+15] (over GGH13 [GGH13a] maps). There exist two sequences of plaintexts $m_1^0 < \cdots < m_\ell^{(0)}$ and $m_1^{(1)} < \cdots < m_\ell^{(1)}$ such that $\mathcal{E}(m_1^0), \cdots, \mathcal{E}(m_\ell^0)$ and $\mathcal{E}(m_1^1), \cdots, \mathcal{E}(m_\ell^1)$ can be efficiently distinguished in the abstract attack model described in Section 2.*

We also show how to extend our attack to obtain an explicit attack when instantiated over GGH13 maps. This attack has an analogous heuristic component as in our attack on obfuscation.

1.3 Defenses

The investigation of the mathematics needed to build iO remains in its infancy. In particular, our work initiates the study of annihilation attacks, and significant future study is needed to see how such attacks can be avoided. We begin that process here with a brief discussion of some promising directions.

As noted above, the primary obstacle to mounting annihilation attacks is finding an efficiently representable annihilating polynomial. Indeed, at present we only know how to mount our attack for a small class of matrix branching programs. Many iO candidates work by first transforming a general program into one of a very specific class of branching programs, that does not include any of the matrix branching programs that we know how to attack using annihilation attacks. However, exactly which iO candidates can be attacked using annihilation attacks, and how annihilation attacks can be prevented, remains unclear.

Is it possible that more complex algebraic constructions of iO candidates can avoid the existence of such annihilating polynomials? For example, we do not know how to extend our attack to the original iO candidate of [GGH+13b], and it is still a possibility that their candidate is secure. The difficulty of extending our attack to their scheme stems from the extra defenses they apply, namely appending random elements to the diagonal of the branching program. This randomization of the branching program means our polynomials do not annihilate, and has so far prevented us from pruning the search space of polynomials to find new annihilating polynomials.

On the other hand, there may still be efficiently computable annihilating polynomials for [GGH+13b], or for that matter any other candidate iO scheme. Given any candidate, how would we argue that no such annihilating polynomials exist? As one approach, we propose exploring ideas from the proof of [Kay09] that shows the existence of sets of polynomials for which no efficient annihilating polynomial can be found, unless **Co-NP** $\subseteq$ **AM**. Perhaps these ideas can be combined with ideas from [BR14, MSW14] to identify a candidate iO scheme where finding relevant annihilating polynomials will be provably as hard as inverting a one-way function.

Going further, we propose exploring *non-algebraic* methods for randomizing the matrix branching programs being obfuscated, in such a way that this randomization destroys all algebraic descriptions of the $\vec{\alpha}$ values that are being given out in encoded form. For example, suppose a matrix branching program can be randomized (while preserving functionality) using matrices drawn randomly from discrete matrix subgroups, resulting in matrices whose entries cannot be written as $\mathbb{Z}$-linear (or $\mathbb{Z}_p$-linear) polynomials. Then the usual algebraic notion of annihilating polynomials may no longer be capable of yielding an attack.

2 Model Description

We now describe an abstract model for attacks on current multilinear map candidates. There are "hidden" variables $X_1, \ldots, X_n$ for some integer n, $Z_1, \ldots, Z_m$ for another integer m, and g. Then there are "public" variable $Y_1, \ldots, Y_m$, which are set to $Y_i = q_i(\{X_j\}) + gZ_i$ for some polynomials q_i. All variables are defined over a field $\mathbb{F}$.

The adversary is allowed to make two types of queries:

- In a **Type 1** query, the adversary submits a "valid" polynomial p_k on the Y_i. Here "valid" polynomials come from some restricted set of polynomials. These restrictions are those that are enforceable using graded encodings. Next, we consider p as a polynomial of the formal variables X_j, Z_i, g. Write $p_k = p_k^{(0)}(\{X_j\}, \{Z_i\}) + gp_k^{(1)}(\{X_j\}, \{Z_i\}) + g^2 \ldots.$

  If p_k is identically 0, then the adversary receives $\perp$ in return. If $p_k^{(0)}$ is *not* identically 0, then the adversary receive $\perp$ in return. If p_k is *not* 0 but $p_k^{(0)}$ *is* identically 0, then the adversary receives a handle to a new variable W_k, which is set to be $p_k/g = p_k^{(1)}(\{X_j\}, \{Z_i\}) + gp_k^{(3)}(\{X_j\}, \{Z_i\}) + \ldots.$

- In a **Type 2** query, the adversary is allowed to submit *arbitrary* polynomials r with small algebraic circuits on the W_k that it has seen so far. Consider $r(\{W_k\})$ as a polynomial of the variables X_j, Z_i, g, and write $r = r^{(0)}(\{X_j\}, \{Z_i\}) + gr^{(1)}((\{X_j\}, \{Z_i\})) + g^2 \ldots.$ If $r^{(0)}$ is identically zero, then the model responds with 0. Otherwise the model responds with 1.

In current graded encoding schemes, the set of "valid" polynomials is determined by the restrictions placed by the underlying set structure of the graded encoding. Here we consider a more abstract setting where the set of "valid" polynomials is arbitrary.

In the standard abstract model for graded encodings, **Type 1** queries output a bit as opposed to an algebraic element, and there are no **Type 2** queries. However, this model has been shown to improperly characterize the information received from **Type 1** queries in current candidate graded encoding schemes. The more refined model above more accurately captures the types of attacks that can be carried out on current graded encodings.

2.1 Obfuscation in the Abstract Model

We now describe an abstract obfuscation scheme that encompass the schemes of [AGIS14, BMSZ15], and can also be easily extended to incorporate the scheme of [BGK+14]. The obfuscator takes as input a branching program of length ℓ, input length n, and arity d. The branching program contains an input function $\mathsf{inp} : [\ell] \to 2^{[n]}$ such that $|\mathsf{inp}(i)| = d$ for all $i \in [\ell]$. Moreover, the branching program contains $2^d\ell + 2$ matrices $A_0, \{A_{i,S_i}\}_{i\in[\ell]}, A_{\ell+1}$ where S_i ranges over subsets of $\mathsf{inp}(i)$, and $A_0 A_{\ell+1}$ are the "bookend" vectors. To evaluate a branching program on input x, we associate x with the set $T \subseteq [n]$ where $i \in T$ if and only if $x_i = 1$. To evaluate the branching program on input x (set T) compute the following product.

$$\mathbf{A}(T) = A_0 \times \prod_{i=1}^{\ell} A_{i,T\cap\mathsf{inp}(i)} \times A_{\ell+1}$$

The output of the branching program is 0 if and only if $\mathbf{A}(T) = 0$.

The obfuscator first generates random matrices $\{R_i\}_{i\in[\ell+1]}$ and random scalars $\{\alpha_{i,S_i}\}_{i\in[\ell],S_i\subseteq\mathsf{inp}(i)}$. Then it computes the randomized branching program consisting of the matrices $\widetilde{A_{i,S_i}} = \alpha_{i,S_i}(R_i \cdot A_{i,S_i} \cdot R_{i+1}^{adj})$ and bookend vectors $\widetilde{A_0} \cdot R_1^{adj}$ and $\widehat{A_{\ell+1}} = R_{\ell+1} \cdot A_{\ell+1}$. It is easy to see that this program computes the same function as the original branching program.

Finally, the obfuscator sets the "hidden" variables in the model to the $\widetilde{A}$ matrices. Denote the "public" variables as $Y_{i,S} = \widetilde{A_{i,S}} + gZ_{i,S} = \alpha_{i,S} R_i \cdot A_{i,S} \cdot R_{i+1}^{adj} + gZ_{i,S}$. The set of valid **Type 1** polynomials is set up so that honest evaluations of the branching program are considered valid. That is, the polynomials

$$p_T = Y_0 \times \prod_{i=1}^{\ell} Y_{i,T\cap\mathsf{inp}(i)} \times Y_{\ell+1}$$

are explicitly allowed. Notice that the g^0 coefficient of p_T is exactly $\widetilde{\mathbf{A}}(\mathbf{T}) \equiv \mathbf{A}(T)$, so the evaluator can run the program by querying on p_T, and checking if the result is $\perp$.

In the case $d = 1$, this obfuscator corresponds to the basic single-input branching program obfuscator of [AGIS14, BMSZ15]. In the more restricted model where there are no **Type 2** queries, it was shown how to set the underlying graded encodings so that the only valid **Type 1** queries are linear combinations of arbitrarily-many p_T polynomials. This is sufficient for indistinguishability obfuscation. When $d = 2$ this corresponds to the dual-input version of these obfuscators, in which it was shown how to set up the underlying graded encodings so that the the linear combination has polynomial size. This is sufficient for virtual black box obfuscation (again in the more restricted model).

In any case, since for functionality the set of allowed queries *must* include honest executions of the program, we always allow queries on the p_T polynomials themselves. As such, our attacks will work by only making **Type 1** queries on honest evaluations of p_T. Thus with *any* restrictions in our abstract model that allow for such honest evaluations of p_T, we will demonstrate how to to break indistinguishability security.

An equivalent formulation. When the obfuscator described above is concretely implemented, the final step is to encode each element of each Y matrix in the GGH13 candidate multilinear map scheme [GGH13a]. Recall that for this, an element $a \in \mathbb{Z}_p$ is mapped to a polynomial $a + gr \in \mathbb{Z}[x]/(x^n + 1)$ (here we omit the level of the encodings, which is without loss of generality since we only compute honest evaluations of the branching program). Then, when evaluating on an input whose output is 0, the g^0-coefficient in p_T will be 0 in the GGH13 ring, namely it will be 0 *modulo the ideal* $\langle g \rangle$. However, we would like this coefficient to be identically 0.

To this end, we note that the obfuscation procedure can be viewed in a slightly different way that will guarantee this. Namely, we *first* encode the A matrices in the GGH13 ring, and *then* we perform the randomization steps over this ring. We note that, crucially, the necessary adjoint matrices R_i^{adj} can be computed over this ring. Then by the properties of the adjoint, we are guaranteed that the off-diagonal entries of each $(R_i^{adj} \times R_i)$ are identically 0, and this ensures that the g^0-coefficient of p_T is as well.

3 Abstract Attack

Here we describe an abstract attack on obfuscation in our generic model. For simplicity, we describe the attack for *single input* branching programs, which proves Theorem 1.1. We extend to dual-input and more generally d-input branching programs in Section 5, which proves Theorem 1.2.

3.1 The branching programs

The first branching program $\mathbf{A}$ is defined as follows. It has $2n+2$ layers, where the first and last layers consist of the row vector $A_0 := (0\ 1)$ and the column vector $A_{2n+1} := (1\ 0)^T$ respectively. The middle $2n$ layers scan through the input bits twice, once forward and once in reverse, with input selection function $\mathsf{inp}(i) := \min(i, 2n+1-i)$ (so x_1 is read in layers 1 and $2n$, x_2 is read in layers 2 and $2n-1$, etc.)[1]. In each of these layers, both matrices are the identity, i.e. we have

$$A_{i,0} = A_{i,1} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

for $i \in [2n]$. Here, we adopt the more standard notation for branching programs where the matrix $A_{i,b}$ is selected if $x_{\mathsf{inp}(i)} = b$.

The branching program $\mathbf{A} = \{\mathsf{inp}, A_0, A_{2n+1}, A_{i,b} \mid i \in [2n], b \in \{0,1\}\}$ is evaluated in the usual way:

$$\mathbf{A}(x) := A_0 \times \prod_{i=1}^{2n} A_{i,x_{\mathsf{inp}(i)}} \times A_{2n+1}.$$

Clearly this satisfies $\mathbf{A}(x) = 0$ for all x.

The second branching program $\mathbf{A}' = \{\mathsf{inp}', A'_0, A'_{2n+1}, A'_{i,b} \mid i \in [2n], b \in \{0,1\}\}$ is defined almost identically. The sole difference is that, in the layers reading bits any of the bits $x_1, \dots, x_k$ for some integer $k \le n$, the matrices corresponding to "$x_i = 0$" are changed to be anti-diagonal. Namely, we have

$$A'_{i,0} = A'_{2n+1-i,0} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \text{ for } i \in [k]$$

and all other components remain the same (i.e. $\mathsf{inp}' = \mathsf{inp}$, $A'_0 = A_0$, $A'_{2n+1} = A_{2n+1}$, and $A'_{i,b} = A_{i,b}$ for all (i,b) where $b = 1$ *or* $i \in [k+1, 2n-k]$). We again have $\mathbf{A}'(x) = 0$ for all x, because the anti-diagonal matrix above is its own inverse and all the matrices commute.

[1] Recall that in the single-input case, the set outputted by $\mathsf{inp}(i)$ is just a singleton set.

3.2 The distinguishing attack

We now specialize the abstract obfuscation scheme from Section 2 to the single-input case. We choose invertible matrices $\{R_i \in \mathbb{Z}_p^{2\times 2}\}_{i\in[2n+1]}$ and non-zero scalars $\{\alpha_{i,x} \in \mathbb{Z}_p\}_{i\in[2n],b\in\{0,1\}}$ uniformly at random. Next, we define

$$\widetilde{A}_0 := A_0 \cdot R_1^{adj} \qquad \widetilde{A}_{2n+2} := R_{2n+1} \cdot A_{2n+1} \qquad \widetilde{A}_{i,b} := \alpha_{i,b} R_i \cdot A_{i,b} \cdot R_{i+1}^{adj}$$

for $i \in [2n]$, $b \in \{0,1\}$, where R_i^{adj} is the adjugate matrix of R_i. Finally, each of the entries of the various $\widetilde{A}$ are what are actually encoded, meaning the "public" variables consist of

$$Y_{i,b} = \alpha_{i,b} R_i \cdot A_{i,b} \cdot R_{i+1}^{adj} + gZ_{i,b}$$

Next, by performing a change of variables on the $Z_{i,b}$, we can actually write

$$Y_{i,b} = \alpha_{i,b} R_i \cdot (A_{i,b} + gZ_{i,b}) \cdot R_{i+1}^{adj}$$

The underlying graded encodings guarantee some restrictions on the types of **Type 1** encodings allowed — however, the restrictions *must* allow evaluation of the branching program on various inputs. In particular, the query

$$p_x := Y_0 \times \prod_{i=1}^{2n} Y_{i,x_{\mathsf{inp}(i)}} \times Y_{2n+1}$$

is allowed. Now, the coefficient of g^0 in p_x is given by

$$p_x^{(0)} := \widetilde{A_0} \times \prod_{i=1}^{2n} \widetilde{A_{i,x_{\mathsf{inp}(i)}}} \times \widetilde{A_{2n+1}} = \rho \prod_i \alpha_{i,x_{\mathsf{inp}(i)}} A_0 \times \prod_{i=1}^{2n} A_{i,x_{\mathsf{inp}(i)}} \times A_{2n+1}$$

which evaluates to 0 by our choice of branching programs. (Note that by the discussion at the end of Section 2, we can take this coeffficient to be identically 0, and not merely divisible by g.) Here $\rho := \prod_i \det(R_i)$ satisfies $\rho I = \prod_i R_i R_i^{adj}$, and we abuse notation by letting $Y_{0,x_{\mathsf{inp}(0)}}$ denote Y_0 (and similarly for the other matrices).

Thus, the model, on **Type 1** query p_x, will return a handle to the variable

$$p_x^{(1)} := \rho \prod_i \alpha_{i,x_{\mathsf{inp}(i)}} \sum_{i=1}^{2n} \Big(\cdots A_{i-1,x_{\mathsf{inp}(i-1)}} \cdot Z_{i,x_{\mathsf{inp}(i)}} \cdot A_{i+1,x_{\mathsf{inp}(i+1)}} \cdots \Big)$$

As in Section 2, we will associate $x \in \{0,1\}^n$ with sets $T \subset [n]$ where $i \in T$ if and only if $x_i = 1$. For $i \in [2,n]$, write $\alpha'_{i,b} = \alpha_{i,b}\alpha_{2n+1-i,b}$. Also set $\alpha'_{1,b} = \rho\alpha_{1,b}\alpha_{2n,b}$. Thus $\rho \prod_i \alpha_{i,x_{\mathsf{inp}(i)}} = \prod_i \alpha'_{i,x_i}$. Define this quantity as $U_x = U_T$. It is straightforward to show that the U_T satisfy the following equation[2] when $|T| \geq 2$.

$$U_T = \prod_{S\subseteq T, |S|\leq 1} U_S^{\left[\binom{|T|-|S|-1}{1-|S|}(-1)^{|S|+1}\right]} = U_\emptyset^{-(|T|-1)} \cdot \prod_{j\in T} U_{\{j\}}$$

Moreover, any equation satisfied by the U_T is generated by these equations.

For the other part of $p_x^{(1)} = p_T^{(1)}$, there are two cases:

[2]See Section 5 for a proof of a more general case.

- The branching program is all-identity. Then $A_{i,0} = A_{i,1} =: A_i$. Here, we write $\beta_{i,b} = \cdots A_{i-1} \cdot Z_{i,x_{\mathsf{inp}(i)}} \cdot A_{i+1} \cdots$. Notice that the $\beta_{i,b}$ are all independent. For $i \in [n]$, let $\beta'_{i,b} = \beta_{i,b} + \beta_{2n+1-i,b}$. Thus,
$$\sum_{i=1}^{2n} \left(\cdots A_{i-1,x_{\mathsf{inp}(i-1)}} \cdot Z_{i,x_{\mathsf{inp}(i)}} \cdot A_{i+1,x_{\mathsf{inp}(i+1)}} \cdots \right) = \sum_{i=1}^{n} \beta'_{i,x_i}$$
Define this quantity as $V_x = V_T$. It is straightforward to show that the V_T satisfy the following equation when $|T| \geq 2$.
$$V_T = \sum_{S \subseteq T, |S| \leq 1} \left[\binom{|T| - |S| - 1}{1 - |S|} (-1)^{|S|+1} \right] V_S = -(|T| - 1)V_\emptyset + \sum_{j \in T} V_{\{j\}}$$
Moreover, any equation satisfied by the V_T is generated by these equations.

  Piecing together, we have that $p_T^{(1)} = U_T V_T$, where U_T, V_T satisfy the equations above.

- The branching program has reverse diagonals for $b = 0$, $i \leq k$. Consider a term $\cdots A_{i-1,x_{\mathsf{inp}(i-1)}} \cdot Z_{i,x_{\mathsf{inp}(i)}} \cdot A_{i+1,x_{\mathsf{inp}(i+1)}} \cdots$. Suppose for the moment that $i \leq k+1$. Since each $A_{i,b}$ is either diagonal or anti-diagonal, we have that $\cdots A_{i-1,x_{\mathsf{inp}(i-1)}} = \cdots A_{i-1,x_{i-1}}$ is equal to the row vector $(0 \;\; 1)$ if the parity of $x_{[1,i-1]}$ is zero, and is equal to $(1 \;\; 0)$ if the parity is 1. Similarly, $A_{i+1,x_{\mathsf{inp}(i+1)}} \cdots$ is equal to the column vector $(1 \;\; 0)^T$ if the parity of $x_{[1,i-1]}$ is zero, and $(0 \;\; 1)^T$ otherwise[3]. Therefore, $\cdots A_{i-1,x_{\mathsf{inp}(i-1)}} \cdot Z_{i,x_{\mathsf{inp}(i)}} \cdot A_{i+1,x_{\mathsf{inp}(i+1)}} \cdots$ is equal to $\left(Z_{i,x_{\mathsf{inp}(i)}}\right)_{1,2}$ or $\left(Z_{i,x_{\mathsf{inp}(i)}}\right)_{2,1}$, depending on the parity of $x_{[1,i-1]}$. Therefore, define $\gamma_{i,b,p}$ to be the result of the product when $x_i = b$ and the parity of $x_{[1,i-1]}$ is p. For $i \in [2n+k, 2n]$, the same holds, so we can absorb the product for this i into $\gamma_{i,b,p}$. For $i \in [k+2, 2n-k-1]$, the same holds true, except that it is only the parity of the bits $x_{[1,k]}$ that matter. Therefore, we can write the product as $\gamma_{i,b,p}$ where $x_i = b$ and the parity of $x_{[1,k]}$ is p. Notice that each of the $\gamma_{i,b,p}$ are independent.

  Define
$$W_T = W_x = \sum_{i=1}^{n} \gamma_{i,x_i,\text{parity}(x_{[1,\min(i-1,k)]})}$$
  Then we have that $p_T^{(1)} = U_T W_T$.

  The W_T must satisfy *some* linear relationships, since the number of W is 2^n, but the number of γ is $4n$. We have not derived a general equation, but instead we will focus on two cases. If the bits $x_1, \ldots, x_k$ are fixed (say to 0), then the parity for these bits is always the same (0). Therefore, W_T for these T satisfy the same equations as the V_T. Thus, any equation satisfied by the $p_T^{(1)}$ for these T in the all-identity case will also be satisfied in the anti-diagonal case.

  In the other case, take $T \subseteq \{1, 2, 3\}$, and suppose $k = 1$. In this simple case, it is straightforward to show that the following are the only linear relationships among these W:
$$W_{1,2,3} + W_1 = W_{1,2} + W_{1,3}$$
$$W_{2,3} + W_\emptyset = W_2 + W_3$$

[3] The rest of the bits of x do not matter, since both matrices for each of these bits occur in the product $A_{i+1,x_{\mathsf{inp}(i+1)}} \cdots$, and therefore cancel out.

These are different, and fewer, than the equations satisfied by the V_T. This will be the basis for our distinguishing attack.

To distinguish the two branching programs, it suffices to find a polynomial Q that annihilates the $p_T^{(1)}$ for $T \subseteq \{1,2,3\}$ in the all-identity case, but does not annihilate in the anti-identity case. Here is such a polynomial:

$$\begin{aligned} Q_{1,2,3} = &\left(p_\emptyset^{(1)} p_{1,2,3}^{(1)}\right)^2 + \left(p_1^{(1)} p_{2,3}^{(1)}\right)^2 + \left(p_2^{(1)} p_{1,3}^{(1)}\right)^2 + \left(p_3^{(1)} p_{1,2}^{(1)}\right)^2 \\ &- 2\left(p_\emptyset^{(1)} p_{1,2,3}^{(1)} p_1^{(1)} p_{2,3}^{(1)} + p_\emptyset^{(1)} p_{1,2,3}^{(1)} p_2^{(1)} p_{1,3}^{(1)} + p_\emptyset^{(1)} p_{1,2,3}^{(1)} p_3^{(1)} p_{1,2}^{(1)} \right. \\ &\qquad \left. + p_1^{(1)} p_{2,3}^{(1)} p_2^{(1)} p_{1,3}^{(1)} + p_1^{(1)} p_{2,3}^{(1)} p_3^{(1)} p_{1,2}^{(1)} + p_2^{(1)} p_{1,3}^{(1)} p_3^{(1)} p_{1,2}^{(1)}\right) \\ &+ 4(p_\emptyset^{(1)} p_{1,2}^{(1)} p_{1,3}^{(1)} p_{2,3}^{(1)} + p_{1,2,3}^{(1)} p_1^{(1)} p_2^{(1)} p_3^{(1)}) \end{aligned}$$

The fact that $Q_{1,2,3}$ annihilates in the all-identity case can be verified by tedious computation. The fact that it does *not* annihilate in the anti-diagonal case can also be verified by tedious computation as follows. Consider a generic degree 4 polynomial Q in the $p_T^{(1)}$ for $T \subseteq \{1,2,3\}$. The condition "Q annihilates the $p_T^{(1)}$" can be expressed as a linear equation in the coefficients of Q. Since Q has degree 4 in 8 variables, the number of coefficients is bounded by a constant, so the linear constraints can be solved. The result of of this computation is that $Q = 0$ is the only solution.

By Schwartz-Zippel, if Q does not annihilate, then with overwhelming probability over the randomness of the obfuscation, the result of applying Q is non-zero.

The attack thus works as follows. First query on inputs x which are zero in every location except the first three bits. Since the branching program always evaluates to zero, the model will return a handle to the element $p_T^{(1)}$, where $T \subseteq \{1,2,3\}$ is the set of bits where x is 1. Then, evaluate the polynomial $Q_{1,2,3}$ on the elements obtained. If the result is 0, then guess that we are in the all-identity case. If the result is non-zero, then guess that we are in the anti-diagonal case. As we have shown, this attack distinguishes the two cases with overwhelming probability.

We make one final observation that will be relevant for attacking the specific [GGH13a] candidate. We note that, for either branching program, the following is true. Let T_0 be some subset of $[k+1, n]$ of size 3, and write $T_0 = i_1, i_2, i_3$. Let T_1 some subset of $[1, n] \setminus T_0$. Then for any subset $T \subseteq [3]$, write $\hat{p}_T^{(1)} := p_{T'}^{(1)}$, where $T' = \{i : i \in T_1 \text{ or } i = i_j \text{ for some } j \in T\}$. If we then evaluate the above polynomial $Q_{1,2,3}$ over the $\hat{p}_T^{(1)}$, we see that it annihilates. This is because the corresponding $p_{T'}^{(1)}$ satisfy the same equations as above.

3.3 Extensions

Here we consider the extension of our attack to other settings.

More general branching programs. First, a straightforward extension of our analysis above shows that $Q_{1,2,3}$ will successfully annihilate for *any* "trivial" branching program where, for each layer i, the matrices $A_{i,0}$ and $A_{i,1}$ are the same. In other words, the evaluation of the branching program is completely independent of the input bits. In contrast, above we showed a very simple branching program which does not satisfy this property for which $Q_{1,2,3}$ does not annihilate.

More generally, it appears that for more complicated branching programs, $Q_{1,2,3}$ will typically annihilate. Therefore, our attack generalizes to distinguish "trivial" branching programs from many complicated branching programs.

Padded-[BMSZ15]. Next, we observe that our attack does not require the branching programs to compute the all-0s function, and that essentially any desired functionality can be used.

Assume that we are given BPs A, A' that both compute the same function $f : \{0,1\}^n \to \{0,1\}$. We augment them to obtain new BPs B, B' by adding 6 extra "padding" layers anywhere in the program; the first two of these input layers read (new) input bit x_{n+1}, the next two read x_{n+2}, and the final two read x_{n+3}. For B we put the identity matrix everywhere in these layers, while for B' we put the anti-identity matrix when the bit read is 0.

The augmented BPs compute essentially the same function as before, namely $f' : \{0,1\}^{n+3} \to \{0,1\}$ where $f'(x) = f(x|_{1\cdots n})$. So, provided it is easy to find an input x for which the original function $f(x) = 0$, we can obtain the outputs $B(x \circ y)$ and $B'(x \circ y)$ for every $y \in \{0,1\}^3$ and evaluate the annihilating polynomial $Q_{1,2,3}$ on them. By the same analysis, this will distinguish the two BPs in our attack model.

[BGK$^+$14] and [BR14]. Our attack also extends to the candidate obfuscator from [BGK$^+$14]. This obfuscator differs slightly from the one described in Section 2, as we describe now. Assume that we start with a BP consisting solely of $w \times w$ matrices (i.e. without bookends), such that the product matrix = identity iff the function evaluates to 0. The [BGK$^+$14] obfuscator first chooses random vectors $s, t \in \mathbb{Z}_p^w$, and adds these as the bookends. Then, in addition to giving out the encoded matrices $Y_{i,b}$ and bookends s and t, the obfuscator gives out the encoded value of the inner product $\langle s, t\rangle$, as well an encoding of each $\alpha_{i,b}$. Finally, evaluation of the obfuscated program on input x is given by

$$s \times \prod_i Y_{i,x_{\mathsf{inp}(i)}} \times t \;\; - \;\; \langle s, t\rangle \cdot \prod_i \alpha_{i,x_{\mathsf{inp}(i)}}$$

which is an encoding of 0 iff the product of the original BP matrices = identity.

Note that the first term in the subtraction matches the polynomial p_x that was analyzed above, so to extend our attack to the [BGK$^+$14] obfuscator we must account for the g^1-coefficient of the second term $\langle s, t\rangle \cdot \prod_i \alpha_{i,x_{\mathsf{inp}(i)}}$. Denoting $\alpha_{0,x_{\mathsf{inp}(0)}} := \langle s, t\rangle$, the polynomial we need to analyze becomes $\prod_i (\alpha_{i,x_{\mathsf{inp}(i)}} + g \cdot z_{i,x_{\mathsf{inp}(i)}})$, where as above the z variables represent the GGH13 randomness. The g^1-coefficient of this polynomial is $\sum_i \left(z_{i,x_{\mathsf{inp}(i)}} \prod_{j \neq i} \alpha_{j,x_{\mathsf{inp}(j)}} \right)$, which we can rewrite as

$$\prod_i \alpha_{i,x_{\mathsf{inp}(i)}} \sum_i \tilde{z}_{i,x_{\mathsf{inp}}(i)} \tag{3.1}$$

via the change of variables $\tilde{z}_{i,x_{\mathsf{inp}}(i)} = z_{i,x_{\mathsf{inp}}(i)} / \alpha_{i,x_{\mathsf{inp}}(i)}$. Finally, observe that expression (3.1) can be easily absorbed into the previous decomposition

$$p_x^{(1)} = \rho \prod_i \alpha_{i,x_{\mathsf{inp}(i)}} \sum_i \beta'_{i,x_i}$$

and indeed the same annihilating polynomial $Q_{1,2,3}$ works for the [BGK$^+$14] obfuscator as well.

We also believe that our attack extends to the candidate obfuscator in [BR14], because evaluating the program in that setting corresponds to a subtraction similar to the one just analyzed. However, we have not completely verified this due to the complexity of the [BR14] construction.

[PST14]. The main difference between the candidate obfuscator in [PST14] and the one in Section 2 is that the initial BP matrices are first padded with extra 1s along the diagonal, i.e. they transform

$$A_{i,b} \;\mapsto\; \begin{pmatrix} A_{i,b} & \\ & I \end{pmatrix}$$

However, since this preserves the property of a layer having the same matrix for both bits, our analysis can be applied to attack this candidate as well.

[GGH⁺13b]. The only candidate branching program obfuscator to which we do not know how to apply our attack is the original candidate due to [GGH⁺13b]. In this candidate, the initial BP matrices are first padded with extra *random* elements on the diagonal, i.e. they transform

$$A_{i,b} \;\mapsto\; \begin{pmatrix} A_{i,b} & \\ & D_{i,b} \end{pmatrix}$$

where $D_{i,b}$ is a random diagonal matrix of dimension d (and $A_{i,b}$ is assumed to have dimension 5). Then, bookend vectors s and t are chosen

$$s = (s_1, \ldots, s_5, \underbrace{0, \ldots, 0}_{d/2}, \underbrace{\$, \ldots, \$}_{d/2}) \qquad t = (t_1, \ldots, t_5, \underbrace{\$, \ldots, \$}_{d/2}, \underbrace{0, \ldots, 0}_{d/2})$$

subject to $\sum_{i \le 5} s_i t_i = 0$. (This is a slight simplification of [GGH⁺13b], but it illustrates the core technical problem in applying our attack.)

Now consider the evaluation on input x:

$$p_x \;=\; s \times \prod_i \begin{pmatrix} A_{i,x_{\mathsf{inp}(i)}} & \\ & D_{i,x_{\mathsf{inp}(i)}} \end{pmatrix} \times t \;=\; s \times \begin{pmatrix} \prod_i A_{i,x_{\mathsf{inp}(i)}} & \\ & \prod_i D_{i,x_{\mathsf{inp}(i)}} \end{pmatrix} \times t$$

While this product indeed encodes value $\sum_{i \le 5} s_i t_i = 0$ when $\prod_i A_{i,x_{\mathsf{inp}(i)}} = I$, the g^1-coefficient becomes quite complicated. This is due to the uniform entries in s and t, which select some of the GGH randomization variables from the "southwest" quadrant of the product matrix. As a result, we do not know how to extend our attack to this setting.

4 Attack on GGH13 Encodings

In this section we explain how the abstract attack above extends to actual obfuscation schemes [BGK⁺14, AGIS14, BMSZ15] when implemented with [GGH13a] multilinear maps. At a high level, this is done by implementing **Type 1** and **Type 2** queries ourselves, without the help of the abstract model's oracle.

Implementing **Type 1** queries is straightforward: for any honestly executed 0-output of the program, namely an encoding

$$p_x = \left[\left(p_x^{(0)}(\{X_j\}, \{Z_i\}) + g p_x^{(1)}(\{X_j\}, \{Z_i\}) + g^2 \ldots \right) / z^k \right]_q$$

with $p_x^{(0)}(\{X_j\}, \{Z_i\}) = 0$, we can multiply by the zero-testing parameter $\mathbf{p}_{\mathsf{zt}} = [hz^k/g]_q$ to obtain

$$W_x := [p_x \cdot \mathbf{p}_{\mathsf{zt}}]_q = h \cdot \left(p_x^{(1)}(\{X_j\}, \{Z_i\}) + g p_x^{(2)}(\{X_j\}, \{Z_i\}) + g^2 \ldots \right) \tag{4.1}$$

This differs from what is returned in the abstract attack because of the factor h. To handle this, we ensure that our annihilating polynomial Q is *homogeneous*, and thus $Q(\{h \cdot p_x^{(1)}(\{X_j\}, \{Z_i\})\}_x) = 0$ whenever $Q(\{p_x^{(1)}(\{X_j\}, \{Z_i\})\}_x) = 0$. (Lemma 5.3 in fact shows we can assume Q is homogeneous without loss of generality, because the $p_x^{(1)}$ are all homogeneous and of the same degree.)

To implement **Type 2** queries, we must check whether a given polynomial Q over $\{W_x\}_{x \in S}$ (for some $S \subseteq \{0,1\}^n$) is an annihilating polynomial, i.e. whether $Q\left(\{h \cdot p_x^{(1)}(\{X_j\}, \{Z_i\})\}_{x \in S}\right) = 0$. To do this we observe that, for any such Q, $Q\left(\{W_x\}_{x \in S}\right)$ produces a ring element in the ideal $\langle hg \rangle$. So, we compute many such elements $v_i = Q_i\left(\{W_x\}_{x \in S_i}\right)$, where Q_i is the (homogeneous) polynomial that annihilates $\{p_x^{(1)}(\{X_j\}, \{Z_i\})\}_{x \in S_i}$ when the encodings were formed by obfuscating the all-identity branching program. More specifically, we compute enough v_i to (heuristically) form a basis of $\langle hg \rangle$. Then, we compute one more element v^* which is either in $\langle hg \rangle$ or not depending on which branching program was obfuscated, and finally we use the $\langle hg \rangle$-basis to test this.

4.1 The Attack

We use essentially the same pair of branching programs $\mathbf{A}, \mathbf{A}'$ that were used in the abstract attack (see Section 3.1): $\mathbf{A}$ consists of all identity matrices, while in $\mathbf{A}'$ the two matrices corresponding to $x_1 = 0$ are changed to be anti-diagonal.

Let $\mathcal{O}$ denote the obfuscator described in Section 2.1. This obfuscator is exactly the one from [BMSZ15], with two exceptions. First, it operates on a branching program reading only one bit per layer, while in [BMSZ15] the branching programs read two bits per layer. In Section 5, we show that our abstract attack, and thus also the concrete attack described here, extends to the dual-input setting. (In fact, we show that it extends to arity-d branching programs for any constant d.) Second, equation (4.1) (and the presence of z^k in $\mathbf{p}_{\mathsf{zt}}$) assumes that all encodings output by $\mathcal{O}$ are level-1 GGH encodings, while in [BMSZ15] a more complicated level structure is used (following [BGK+14, MSW14]). However, since our attack only uses these encodings to honestly execute the obfuscated program, (4.1) holds even for this level structure.

Here is our attack:

- Let $m = n^{O(1)}$ be the dimension of the underlying encodings (this is a parameter of the [GGH13a] scheme). Note that any m linearly independent elements of $\langle hg \rangle$ form a basis for $\langle hg \rangle$. Let $m' \gg m$ be an integer.
- Repeat the following for $t = 1, \ldots, m'$:
  - Choose a random size-3 subset $T_0 = \{i_1, i_2, i_3\} \subseteq [n]$ that *does not* contain 1. T_0 will correspond to the set of input bits that we vary.
  - Choose a random subset $T_1 \subseteq ([n] \setminus T_0)$. T_1 will correspond to a fixing of the bits outside T_0.
  - For each $T \subseteq [3]$,
    - ∗ let $x_T \in \{0,1\}^n$ be the string such that $x_i = 1$ if and only if either $i \in T_1$, or $i = i_j$ for some $j \in T$ (recall that $T_0 = \{i_1, i_2, i_3\}$).
    - ∗ Run the obfuscated program on input x, until the zero test query. Let $p_T^{(1)}$ be the vector obtained from zero testing.

– Evaluate the polynomial $Q_{1,2,3}$ in Section 3 on the $p_T^{(1)}$. Let the output be defined as v_t. That is, we let x_T vary over the the 8 possible values obtained by fixing all the input bits outside of T_0, run the obfuscated program on each of the x_T, and then evaluate the polynomial $Q_{1,2,3}$ on the results to get v_t.

- Find a linearly independent subset V of the v_t.
- Choose a random size-3 subset $T_0^* = \{i_1, i_2, i_3\} \subseteq [n]$ that *does* contain 1. For each $T \subseteq [3]$, compute $p_T^{(1)}$ as above. Then evaluate the polynomial $Q_{1,2,3}$ on the $p_T^{(1)}$ to obtain a vector v^*.
- Finally, test if v^* is in the span of V. If it is, output 1. Otherwise, output 0.

Analysis of our attack. As in Section 3, let $T_0 \subseteq [n]$, and choose an arbitrary fixing of the remaining bits. Suppose we evaluate the branching program on the 8 different inputs corresponding to varying the bits in T_0, and then run the polynomial $Q_{1,2,3}$ on the results. Then $Q_{1,2,3}$ annihilates annihilates in either of the following cases:

- T_0 does not contain 1.
- The branching program is the all-identity program, even if T_0 contains 1.

Therefore, we see that $Q_{1,2,3}$ annihilates for each $t = 1, \ldots, m'$. In the case of [GGH13a], $Q_{1,2,3}$ annihilating mans that the resulting vector v is an element of the ideal $\langle hg \rangle$.

Thus, each of the v_t are elements in the ideal, regardless of the branching program. We will heuristically assume that the v_t span the entire ideal. This is plausible since the number m' of v_t is much larger than the dimension of the ideal. Increasing m' relative to m should increase the likelihood of the heuristic being true.

For v^*, however, things are different. v^* is in the ideal if the branching program is the all-identity, but outside the ideal (with high probability) if the branching program has anti-diagonals, since in this case $Q_{1,2,3}$ does not annihilate. Therefore, our test for v^* being linearly independent from v will determine which branching program we were given.

5 Beyond Single-Input Branching Programs

In this section, we show an abstract attack on dual-input branching programs. More generally, we show that generalizing to d-input branching programs for any constant d will not prevent the attack.

We first recall our semantics of branching programs in the general d-ary setting. Fix integers d, ℓ and n which respectively correspond to the number of bits read by each layer of the branching program, the length of the branching program, and the input length. Let $\mathsf{inp} : [\ell] \to 2^{[n]}$ be any function such that $|\mathsf{inp}(i)| = d$ for all $i \in [\ell]$. A branching program of length ℓ then consists of $2^d\ell + 2$ matrices $A_0, \{A_{i,S_i}\}_{i \in [\ell]}, A_{\ell+1}$ where S_i ranges over subsets of $\mathsf{inp}(i)$, and $A_0 A_{\ell+1}$ are the "bookend" vectors.

We associate an input x with the subset $T \in 2^{[n]}$ of indices where x is 1. To evaluate the branching program on input x (set T) compute the product

$$\mathbf{A}(T) = A_0 \times \prod_{i=1}^{\ell} A_{i,T \cap \mathsf{inp}(i)} \times A_{\ell+1}$$

Consider the obfuscation of the branching program. Let R_i be the Kilian randomizing matrices. Let $\alpha_{i,S}$ be the extra randomization terms. Then the encoded values seen by the adversary are the matrices $Y_{i,S} = \alpha_{i,S} R_i . A_{i,S} . R_{i+1}^{adj} + gZ_{i,S}$

By performing a change of variables on the $Z_{i,S}$, we can actually write $Y_{i,S} = \alpha_{i,S} R_i \cdot (A_{i,S} + gZ_{i,S}) \cdot R_{i+1}^{adj}$

The encodings will guarantee some restrictions on the **Type 1** queries allowed — however they must allow evaluation of the branching program. Thus we assume that the following query is allowed for every $T \subseteq [n]$.

$$p_T = Y_0 \times \prod_{i=1}^{\ell} Y_{i,T\cap \mathsf{inp}(i)} \times Y_{\ell+1}$$

Now we will assume a trivial branching program where (1) within each layer, all matrices are the same ($A_{i,S_i} = A_{i,S'_i}$ for any $S_i, S'_i \in \mathsf{inp}(i)$), so in particular the program is constant, and (2), the branching program evaluates to 0 on all inputs. Therefore, the g^0 coefficient in p_T will evaluate to zero everywhere. Thus, a **Type 1** query will output a handle to the variable

$$p_T^{(1)} = \rho \left(\prod_i \alpha_{i,S\cap \mathsf{inp}(i)} \right) \sum_i \left(\cdots A_{i,T\cap \mathsf{inp}(i-1)} \cdot Z_{i,T\cap \mathsf{inp}(i)} \cdot A_{i+1,T\cap \mathsf{inp}(i+1)} \cdots \right)$$

For any sets $S' \subseteq S \subseteq [n]$ with $|S| = d$, define

$$\alpha_{S,S'} := \prod_{i:\mathsf{inp}(i)=S} \alpha_{i,S'} \qquad\qquad \beta_{S,S'} := \sum_{i:\mathsf{inp}(i)=S} \beta_{i,S'}$$

and for any set $T \subseteq [n]$, define

$$U_T := \prod_{S:|S|=d} \alpha_{S,T\cap S} \qquad\qquad V_T := \sum_{S:|S|=d} \beta_{S,T\cap S}$$

Then we have that $p_T^{(1)} = U_T V_T$.

The following theorem shows that, for $|T| > d$, U_T and V_T can each be written as rational polynomials in the variables $U_{T'}, V_{T'}$ for $|T'| \le d$.

Theorem 5.1. *Let $T \subseteq [n]$ with $|T| > d$. Then,*

$$U_T = \prod_{T' \subseteq T : |T'| \le d} U_{T'}^{(-1)^{d-|T'|} \cdot \binom{|T|-|T'|-1}{d-|T'|}}$$

and

$$V_T = \sum_{T' \subseteq T : |T'| \le d} (-1)^{d-|T'|} \cdot \binom{|T| - |T'| - 1}{d - |T'|} \cdot V_{T'}.$$

Proof. We prove this equation for V_T, the proof for U_T is analogous. Consider expanding the left and right sides of the equation in terms of the $\beta_{S,Z}$ and equating the coefficients of $\beta_{S,Z}$ on both sides, we see that the following claim suffices to prove the theorem:

Claim 5.2. *For any sets T, S, Z,*

$$\sum_{T' \subseteq T : |T'| \le d, T' \cap S = Z} \binom{|T| - |T'| - 1}{d - |T'|} (-1)^{d-|T'|} = \begin{cases} 1 & \text{if } T \cap S = Z \\ 0 & \text{if } T \cap S \ne Z \end{cases}$$

The left hand side (resp. right hand side) of the above equation corresponds to the coefficient of $\beta_{S,Z}$ in the right hand side (resp. left hand side) of the V equation in Theorem 5.1. Hence the theorem follows from the claim.

We now prove the claim. First, suppose $Z \nsubseteq T \cap S$. Then the sum on the right is empty, so the result is zero, as desired. Next, suppose $Z \subseteq T \cap S$. Then for any T' in the sum, we can write $T' = Z \cup T''$ where $T'' \subseteq T \setminus (S \cup Z)$ and $|T''| \leq d - |Z|$. Therefore, we can think of the sum as being over T''. The number of T'' of size i is $\binom{|T\setminus(S\cup Z)|}{i}$. Therefore, the sum on the left is equal to

$$\sum_{i=0}^{d-|Z|} \binom{|T \setminus (S \cup Z)|}{i} \binom{|T| - |Z| - i - 1}{d - |Z| - i} (-1)^{d-i-|Z|}$$

Let $e = d - |Z|$, $t = |T| - |Z| = |T \setminus Z|$ (since $Z \subseteq (T \cap S) \subseteq T$), and $k = |T \setminus (S \cup Z)|$. Notice that $k \leq t$, and that $k = t$ if and only $Z = T \cap S$. Thus, we need to show that

$$\sum_{i=0}^{e} \binom{k}{i} \binom{t-i-1}{e-i} (-1)^{e-i} = \begin{cases} 1 & \text{if } k = t \\ 0 & \text{if } k < t \end{cases}$$

First, we use the identity $(-1)^s \binom{s-r-1}{s} = \binom{r}{s}$ with $s = e - i$ and $r = e - t$ to replace $\binom{t-i-1}{e-i}(-1)^{e-i}$ with $\binom{e-t}{e-i}$ (note that the binomial coefficients are defined for negative integers such as $e - t$).

Then we have that the left hand side becomes $\sum_{i=0}^{e} \binom{k}{i}\binom{e-t}{e-i}$. The Chu-Vandermonde identity shows that this is equal to $\binom{k+(e-t)}{e} = \binom{e-(t-k)}{e}$. Notice that if $t = k$, the result is 1. Moreover, if $k < t$, then the upper index of the binomial is less than the bottom index, so the result is 0. This proves the claim and hence the theorem. $\square$

Annihilating polynomial for $p_T^{(1)}$. We now describe our abstract attack using annihilating polynomials. The first step is to argue that it is possible to efficiently devise a non-zero polynomial Q on several of the $p_T^{(1)}$ such that Q is identically zero when the $p_T^{(1)}$ come from the obfuscation. In particular, we need Q to be identically zero as a polynomial over the α's and β's. Using Theorem 5.1, it suffices to find Q that is identically zero as a rational function over the U_T, V_T for $|T| \leq d$.

We will first consider the values $p_T^{(1)}$ as polynomials in the $V_T, U_T, |T| \leq d$ over the rationals. Let $k = 2d+2$, and consider all $p_T^{(1)}$ for $T \subseteq [k]$. Then each $p_T^{(1)}$ is a rational function of the U_T, V_T for $T \subseteq [k], |T| \leq d$. There are $\sum_{i=0}^{d} \binom{k}{i} < 2^{2d+1}$ such T, and therefore fewer than 2^{2d+2} such U_T, V_T. Yet there are 2^{2d+2} different $p_T^{(1)}$ for $T \subseteq [k]$ of arbitrary size. Thus, there must be some algebraic dependence among the $p_T^{(1)}$. Notice moreover that the expression for $p_T^{(1)}, T \subseteq [k]$ in terms of the $U_{T'}, V_{T'}, T' \subseteq [k], |T'| \leq d$ are fixed rational functions with integer coefficients, independent of the branching program, n, or ℓ; the only dependence is on d. Recall that we are taking d to be a constant, so the number of $p_T^{(1)}, V_{T'}, U_{T'}$ and the coefficients in the relation between them are all constants. Therefore, there is a fixed polynomial Q_d in the $p_T^{(1)}$ over the rationals such that Q_d is identically zero when the $p_T^{(1)}$ come from obfuscation.

We note that by a more tedious argument, it is actually possible to show there must be an algebraic dependence among the $p_T^{(1)}$, and hence an annihilating polynomial for them, when T varies over the subsets of $[k]$ for $k = 2d + 1$ (as opposed to $2d + 2$).

By multiplying by the LCM of the denominators of the rational coefficients, we can assume without loss of generality that Q_d has integer coefficients. Therefore, there is a fixed integer polynomial Q_d such that $Q_d(p_T^{(1)})$ is identically 0. Since the coefficients are integers, this polynomial actually also applies in any field or ring; we just need to verify that it is not identically zero in the field/ring. This will be true as long as the characteristic of the ring is larger than the largest of the coefficients. Since in our case, the ring characteristic grows (exponentially) with the security parameter, for high enough security parameter, the polynomial Q_d will be non-zero over the ring.

Computing the annihilating polynomial Q_d. In Section 3, we gave an annihilating polynomial for the case $d = 1$. For more general d, we do not know a more general expression. However, we still argue that such a Q_d can be efficiently found for any d:

- The polynomial Q_d is just a fixed polynomial over the integers; in particular is has a constant-sized description for constant d. Thus, we can assume that Q_d is simply given to the adversary.
- If we want to actually compute Q_d, this is possible using linear algebra. Using degree bounds for the annihilating polynomial due to [Kay09], we can determine an upper bound t on the degree of Q_d. Then, the statement "Q_d annihilates the $p_T^{(1)}$" can be expressed as a system of linear equations in the coefficients of Q_d, where the equations themselves are determined by expressions for $p_T^{(1)}$ in terms of the $U_{T'}, V_{T'}$. By solving this system of linear equations, it is possible to obtain a polynomial Q_d. We note that, for constant d, t will be constant, the system of linear constraints will be constant, and hence it will take constant time to compute Q_d. In terms of d, the running time is necessarily exponential (since the number of variables $p_T^{(1)}$ is exponential).

The following lemma shows that we can take Q to be a homogeneous polynomial, which will be necessary for obtaining an attack over [GGH13a].

Lemma 5.3. *Let $p_1, \dots, p_k$ be homogeneous polynomials each of the same degree d. Let Q be any polynomial that annihilates $\{p_i\}_i$, and let Q_r denote the homogeneous degree-r part of Q. Then Q_r annihilates $\{p_i\}_i$ for each $r \leq \deg(Q)$.*

Proof. If $Q_r(\{p_i\}_i) \neq 0$ for some $r \leq \deg(Q)$, then Q_r contains some degree-dr monomial m. Then because $\sum_{r=0}^{\deg(Q)} Q_r(\{p_i\}_i) = Q(\{p_i\}_i) = 0$, some $Q^{(r')}$ for $r' \neq r$ must contain the monomial $-m$. However, since $Q^{(r')}$ is homogeneous of degree $dr' \neq dr$, this is a contradiction. □

Completing the attack. Using the annihilating polynomial above, we immediately get an attack on the abstract model of obfuscation. The attack distinguishes the trivial branching program where all matrices across each layer are the same, from a more general all-zeros branching program that always outputs zero, but has a non-trivial branching program structure.

The attack proceeds as follows: query the model on **Type 1** queries for all p_T as T ranges over the subsets of $[k]$. Since the branching program always outputs 0, the model will return a handle to the $p_T^{(1)}$ polynomials. Then evaluate the annihilating polynomial Q_d above on the obtained $p_T^{(1)}$. If the result is non-zero (as will be the case for many non-trivial branching programs), then we know the branching program was *not* the trivial branching program. In contrast, if the result *is* zero, then we can safely guess that we are in the trivial branching program case. Hence, we breach the indistinguishability security of the obfuscator.

6 Attacking Order Revealing Encryption

In this section, we describe how to attack the order revealing encryption (ORE) scheme of Boneh et al. [BLR+15]. We first recall the definition of an order revealing encryption scheme.

Definition 6.1. An order revealing encryption scheme consists of four algorithms ($\texttt{Gen}, \texttt{Enc}, \texttt{Dec}, \texttt{Comp}$) such that:

- $\texttt{Gen}$ takes as input the security parameter, and outputs public parameters $\texttt{PP}$ and a secret key $\texttt{sk}$.
- $\texttt{Enc}(\texttt{sk}, m)$ is a secret key encryption algorithm that outputs a ciphertext c.
- $\texttt{Dec}(\texttt{sk}, c)$ is a decryption algorithm that outputs a plaintext.
- $\texttt{Comp}(\texttt{PP}, c_0, c_1)$ is a public key comparison procedure that takes as input two ciphertexts, and outputs a bit b.
- **Correct Decryption**. This is the standard correctness requirement for secret key encryption. For any m, with overwhelming probability over the choice of $(\texttt{PP}, \texttt{sk})$ and the random coins of $\texttt{Enc}$, we have that $\texttt{Dec}(\texttt{sk}, \texttt{Enc}(\texttt{sk}, m))$ outputs m.
- **Correct Comparison**. For any messages m_0, m_1, $m_0 < m_1$, we have that with overwhelming probability over the choice of $(\texttt{PP}, \texttt{sk})$ and the random coins of $\texttt{Enc}$, $\texttt{Comp}(\texttt{PP}, \texttt{Enc}(\texttt{sk}, m_0), \texttt{Enc}(\texttt{sk}, m_1)) = 0$ and $\texttt{Comp}(\texttt{PP}, \texttt{Enc}(\texttt{sk}, m_1), \texttt{Enc}(\texttt{sk}, m_0)) = 1$
- **Security**. For any two polynomial-length sequences of ordered messages $m_0^{(0)} < m_1^{(0)} < \cdots < m_\ell^{(0)}$ and $m_0^{(1)} < m_1^{(1)} < \cdots < m_\ell^{(1)}$ of the same length ℓ, we have that the following two distributions are computationally indistinguishable: $\texttt{PP}, \texttt{Enc}(\texttt{sk}, m_0^{(0)}), \texttt{Enc}(\texttt{sk}, m_1^{(0)}), \ldots, \texttt{Enc}(\texttt{sk}, m_\ell^{(0)})$ and $\texttt{PP}, \texttt{Enc}(\texttt{sk}, m_0^{(1)}), \texttt{Enc}(\texttt{sk}, m_1^{(1)}), \ldots, \texttt{Enc}(\texttt{sk}, m_\ell^{(1)})$

We note that the security definition is much weaker than that defined in [BLR+15], which allowed for adaptive message queries. Nonetheless, we will give an attack on their scheme even for our weaker definition

6.1 Description of [BLR+15] in Abstract Model

We now given an abstract description of the [BLR+15] order revealing encryption scheme in our model for graded encodings. We will actually describe a simplified variant for which, for any ciphertext, that ciphertext can be inserted into either the first or second ciphertext slot of $\texttt{Comp}$, but not both. That is, $\texttt{Enc}$ now takes as input an additional bit b, and if $b = 0$, and $\texttt{Comp}(\texttt{PP}, c_0, c_1)$ is only required to be correct where c_0 is encrypted using bit 0, and c_1 is encrypted using bit 1. This is how the Boneh et al. [BLR+15] protocol works; to obtain the usual notion of order revealing encryption, the encryption procedure simply encrypts twice, once to each input.

The starting point for the construction is a branching program $A_{1,0}, A_{1,1}, B_{1,0}, B_{1,1}, \ldots, A_{n,0}, A_{n,1}, B_{n,0}, B_{n,1}$ such that:

- For any two n-bit integers x, y, $\prod_{i=1}^n A_{x_i} \cdot B_{y_i} = 0$ if and only if $x < y$. Note that it is trivial to extend our attacks to work in the case where $<$ is replaced with $\leq, >$, or $\geq$.

- For any $j \in [n]$, the products $\prod_{i=1}^{j} A_{x_i} \cdot B_{y_i}$ and $\prod_{i=n-j+1}^{n} A_{x_i} \cdot B_{y_i}$, which will be vectors of some dimension, only depend on the result of comparing the first or last, respectively, j bits of x and y. That is, $\prod_{i=1}^{j} A_{x_i} \cdot B_{y_i}$ takes on one of 3 possible values, depending on the three possible results of comparing $x_{[1,i]}, y_{[1,i]}$, $<, >$, or $=$.

Note that here we describe a branching program without bookends, but where the matrices are shaped so that the output is a scalar. It is straightforward to obtain a branching program in this form by multiplying the branching program by the bookend vectors.

The secret key `sk` for the ORE scheme consists of $2n-1$ random matrices R_i, as well as the necessary information to compute encodings in the graded encoding. The public key will be the description of the graded encoding scheme, which allows for **Type 1** and **Type 2** queries, with the class of valid **Type 1** queries to be specified later.

Encryption. To encrypt integer x into the left input to `Comp`, choose random $\alpha_{x,i}$ for $i \in [n]$, and compute $\widetilde{A_{x,i}} = \alpha_{x,i} R_{2i-2} \cdot A_{i,x_i} \cdot R_{2i-1}^{adj}$. Here, R_0 is just the integer 1. Then the $\widetilde{A_{x,i}}$ are encoded, meaning the public values seen by the adversary are

$$X_{x,i} = \alpha_{x,i} R_{2i-2} \cdot (A_{i,x_i} + gZ_{x,i}) \cdot R_{2i-1}^{adj}$$

for random $Z_{x,i}$. Here, we use the re-labeling of the Z variables used in Section 3.

Encryption in the right input to `Comp` is analogous. Choose random $\beta_{x,i}$ for $i \in [n]$, and compute $\widetilde{B_{x,i}} = \beta_{x,i} R_{2i-1} \cdot B_{i,x_i} \cdot R_{2i}^{adj}$. Here, R_{2n} is just the integer 1. Then the $\widetilde{B_{x,i}}$ are encoded, meaning the public values seen by the adversary are

$$Y_{x,i} = \beta_{x,i} R_{2i-1} \cdot (B_{i,x_i} + gW_{x,i}) \cdot R_{2i}^{adj}$$

for random $W_{x,i}$.

Comparison. To compare two ciphertexts c_0, c_1 consisting of $X_{x,i}$ and $Y_{y,i}$ for integers x, y, perform a **Type 1** query on the product

$$\prod_{i=1}^{n} X_{x,i} \cdot Y_{y,i} = \prod_{i=1}^{n} \alpha_{x,i}\beta_{y,i} \left(A_{i,x_i} + gZ_{x,i}\right) \cdot \left(B_{i,y_i} + gW_{y,i}\right)$$

Notice that the g^0 term is exactly equal to $\prod_{i=1}^{n} A_{x_i} \cdot B_{y_i}$, up to scaling by the $\alpha_{x,i}, \beta_{y,i}$. Therefore, the result is zero if and only if $x < y$. Thus, it is possible to determine the order of the two plaintexts.

Note that these **Type 1** queries must be explicitly allowed for correctness. [BLR+15] analyze the types of queries that are allowed in the standard generic model for graded encodings; however, for our attack, we do not require any other **Type 1** queries.

6.2 Our Attack

Suppose `Comp` gives 0 on encryptions of x and y. The coefficient of g^1 will be:

$$V(x,y) = \left(\prod_{i=1}^{n} \alpha_{x,i}\beta_{y,i}\right) \sum_{i=1}^{n} \left(\prod_{j=1}^{i-1} A_{j,x_j} \cdot B_{j,y_j}\right) \cdot \left(A_{i,x_i} \cdot W_{y,i} + Z_{x,i} \cdot B_{i,y_i}\right) \cdot \left(\prod_{j=i+1}^{n} A_{j,x_j} \cdot B_{j,y_j}\right)$$

Define $\alpha_x = \prod_{i=1}^n \alpha_{x,i}$ and $\beta_y = \prod_{i=1}^n \beta_{y,i}$. Recall that $\prod_{j=1}^{i-1} A_{j,x_j} \cdot B_{j,y_j}$ only depends on the result of comparing the first $i-1$ bits, and that $\prod_{j=i+1}^{n} A_{j,x_j} \cdot B_{j,y_j}$ only depends on the result of comparing the last $n-i$ bits. Therefore, we can re-write the g^1 coefficient as:

$$V(x,y) = \alpha_x \beta_y \sum_{i=1}^{n} \Big(Z_{x,i,\texttt{Comp}(x_{[1,i-1]},y_{[1,i]}),y_i,\texttt{Comp}(x_{[i+1,n]},y_{[i+1,n]})} + W_{y,i,\texttt{Comp}(x_{[1,i-1]},y_{[1,i-1]}),x_i,\texttt{Comp}(x_{[i+1,n]},y_{[i+1,n]})} \Big)$$

For variables $Z_{x,i,a,b,c}, W_{y,i,a,b,c}$ where $a, c \in \{<,=,>\}$ and $b \in \{0,1\}$.

Choosing the query points. We now describe how we choose our query points. Let k be a positive integer, and $n = 2k+4$. Let X_0, Y_0, X_1, Y_1 be sets of n-bit integers that have the form:

- X_b: $x = 0\ \hat{x}\ 00\ 0^k\ 0$ for a k-bit integer $\hat{x}$. In particular $X_0 = X_1$.
- Y_b: $y = b\ 1^k\ 11\ \hat{y}\ b$ for a k-bit integer $\hat{y}$.

Then X_b, Y_b satisfy the following:

- For any $x \in X_b, y \in Y_b$, $x < y$.
- For any $x \in X_1, y \in Y_1$, $x_{[1,i]} < y_{[1,i]}$ for all $i \in [n]$. That is, the result of comparing the first i bits for any i is always $<$.
- For any $x \in X_0, y \in Y_0$, $x_{[1,i]} < y_{[1,i]}$, *unless*
  - $i = 1$
  - $i \in [2, k+1]$ and $\hat{x}_{[1,i-1]} = 1^{i-1}$.
- For any $x \in X_1, y \in Y_1$, $x_{[i,n]} < y_{[i,n]}$ for any $i \in [n]$. That is, the result of comparing the last $n-i+1$ bits for any i is always $<$.
- For any $x \in X_0, y \in Y_0$, $x_{[i,n]} < y_{[i,n]}$, *unless*
  - $i = n$
  - $i \in [k+4, 2k+3]$ and $\hat{y}_{[i-k-3,k]} = 1^{i-k-3}$.

We first consider X_1, Y_1. For these x, y, $\texttt{Comp}(x_{[1,i-1]}, y_{[1,i-1]})$ and $\texttt{Comp}(x_{[i+1,n]}, y_{[i+1,n]})$ will always be the independent of the choice of $x \in X_1$ and $y \in Y_1$, namely $<$[4]. Moreover, for $i = 1$ or $i \in [k+2, n]$, x_i is independent of x. Therefore, the $Z_{x,i,\texttt{Comp}(x_{[1,i-1]},y_{[1,i]}),y_i,\texttt{Comp}(x_{[i+1,n]},y_{[i+1,n]})}$ for these i are independent of y. They can thus be absorbed into the other Z_x's. Similar statements hold for the y_i's for $i \in [1, k+3]$ or $i = n$.

This lets us write

$$V(x,y) = \alpha_x \beta_y \sum_{i=1}^{k} (Z_{x,i,\hat{y}_i} + W_{y,i,\hat{x}_i})$$

[4]Technically, in the case $i = 1$, $\texttt{Comp}(x_{[1,i-1]}, y_{[1,i-1]})$ will give $=$. However, this is still independent of the choice of x and y, so all of the following arguments are still valid.

for $x \in X_1$, $y \in Y_1$, and variables $Z_{x,i,b}, W_{y,i,b}$. We write this as the sum of two inner products: $V(x,y) = Z_x \cdot \Gamma_y + \Delta_x \cdot W_y$ where

$$\begin{aligned}
Z_x &= \alpha_x(Z_{x,1,0} \;\; Z_{x,1,1} \;\; Z_{x,2,0} \;\; Z_{x,2,1} \;\; \cdots \;\; Z_{x,n,0} \;\; Z_{x,n,1}) \\
W_y &= \beta_y(W_{y,1,0} \;\; W_{y,1,1} \;\; W_{y,2,0} \;\; W_{y,2,1} \;\; \cdots \;\; W_{y,n,0} \;\; W_{y,n,1}) \\
\Delta_x &= \alpha_x(\;\; (1-\hat{x}_1) \;\; \hat{x}_1 \;\; (1-\hat{x}_2) \;\; \hat{x}_2 \;\; \cdots \;\; (1-\hat{x}_n) \;\; \hat{x}_n \;\;) \\
\Gamma_y &= \beta_y(\;\; (1-\hat{y}_1) \;\; \hat{y}_1 \;\; (1-\hat{y}_2) \;\; \hat{y}_2 \;\; \cdots \;\; (1-\hat{y}_n) \;\; \hat{y}_n \;\;)
\end{aligned}$$

Let V be the matrix of $V(x,y)$ values as x, y vary over X_1, Y_1, respectively. Let Z, Δ be the matrices containing the vectors Z_x, Δ_x (respectively) as rows, and let W, Γ be the matrices containing the vectors W_y, Γ_y (respectively) as columns. Then we can write

$$V = Z \cdot \Gamma + \Delta \cdot W = (Z \;\; \Gamma) \cdot \begin{pmatrix} \Delta \\ W \end{pmatrix}$$

Now, the smallest dimension of the matrices Γ, Δ is $2k$, so their rank is clearly at most $2k$. We now argue that the rank is in fact at most $k+1$ for each. To see this, note that the columns of Δ are spanned by the following $k+1$ column vectors: $v_x^{(0)} = \alpha_x$, and $v_x^{(i)} = \hat{x}_i$ for $i \in [k]$. Thus the rank of Δ is at most $k+1$. Moreover, it is straightforward to argue that any $k' \leq k+1$ rows of Δ are linearly independent. Similar arguments hold for Γ.

Since Z, W are full rank with overwhelming probability, $Z \cdot \Gamma$ and $\Delta \cdot W$ each have rank $\min(k+1, 2^k)$. Therefore, their sum V has rank at most $2k+2$. Moreover, since Z, W are random matrices, the ranks will add with overwhelming probability, so the total rank is $\min(2k+2, 2^k)$.

We now consider X_0, Y_0. Performing a similar treatment as we did in the case of X_1, Y_1, for any $x \in X_0, y \in Y_0$, we can therefore write

$$V(x,y) = \alpha_x \beta_x \left(\left(\sum_{i=1}^{k-1} Z_{x,i,\hat{y}_i,\delta_0(\hat{y}_{[i+1,k]})} \right) + \left(\sum_{i=2}^{k} W_{y,i,\hat{x}_i,\delta_1(\hat{x}_{[1,i-1]})} \right) \right)$$

where $\delta_b(z)$ is 1 if and only if all the bits of z are equal to b, and 0 otherwise. Note that one might expect there to be a $Z_{x,0,\delta_0(\hat{y})}$ term. However, $\delta_0(\hat{y})$ is determined by $\hat{y}_1$ and $\delta_0(\hat{y}_{[2,k]})$, and hence $Z_{x,0,\delta_0(\hat{y})}$ can be absorbed into $Z_{x,1,\hat{y}_1,\delta_0(\hat{y}_{[2,k]})}$. Similar statements hold for $W_{y,k+1,\delta_1(\hat{x})}, Z_{x,k,\hat{y}_k}$, and $W_{y,1,\hat{x}_1}$.

Through a similar analysis as in the X_1, Y_1 case, the matrix V whose entries are $V(x,y)$ can be written as $Z \cdot \Gamma + \Delta \cdot W$ for $2^k \times (4k-4)$ matrices Z, Δ and $(4k-4) \times 2^k$ matrices Γ, W, where Z, W contain the variables $Z_{x,i,b,c}, W_{y,i,b,c}$ and Δ, Γ are matrices that depend on the bits of x, y. Note that these matrices will be different than those computed in the X_1, Y_1 case. The matrices Δ, Γ can each be shown to have rank $\min(2k, 2^k)$. Then V has rank $\min(4k, 2^k)$.

Hence, the rank of V will depend on whether we consider X_0, Y_0 or X_1, Y_1. This will be the basis for our attack.

The attack. We now describe our distinguishing attack. Set $k = 4$, and let X_0, Y_0, X_1, Y_1 be the sets of $2^k = 16$ integers each, as above.

- Query on the sequences (X_0, Y_0) and (X_1, Y_1), obtaining 32 ciphertexts corresponding to encryptions of (X_b, Y_b). Let D be the ciphertexts encrypting X_b, and E be the ciphertexts encrypting Y_b. Note that we only need the ciphertexts in D to be valid *left* inputs to `Comp`, and the ciphertexts in E to be valid *right* inputs.
- For each $d \in D, e \in E$, make a **Type 1** query on the polynomial corresponding to runing the comparison procedure on d, e. Since $x < y$ for each $x \in X_b, y \in Y_b$, the polynomial will evaluate to 0, and hence result of the query will be an algebraic element $V_{d,e}$.
- Assemble the $2^k = 16 \times 2^k = 16$ matrix V of the $V_{d,e}$ components.
- Compute the determinant of V. If the result is zero, output 1. Otherwise, output 0.

In the case $b = 1$, V will have rank $2k + 2 = 10 < 16$. Hence the determinant gives 0. In the vase $b = 0$, V will have rank $4k = 16$. Hence the determinant will be non-zero with overwhelming probability. Thus, our attack successfully determines which set of encryptions it received.

Attack over GGH13. We now describe how to turn this into an actual attack on ORE built on GGH13 multilinear maps. Let ℓ be some integer. Let $X^{(a)}$ be X_1, except with the ℓ-bit integer a prepended to each of the elements in X_1. Similarly define $Y^{(a)}$. Define X_b^*, Y_b^* as X_b, Y_b, except with 0^ℓ prepended to each element.

Let $m = n^{O(1)}$ be the dimension of the underlying encodings, and $m' \gg m$ be an integer. Let $S \subseteq \{0,1\}^\ell$ be a set of size m' that does not contain zero. We will attack ORE instantiated with $(12 + \ell)$-bit integers.

The attack works as follows:

- Query on sequences $(X_0^*, Y_0^*), \{(X^{(s)}, Y^{(s)})\}_{s \in S}$ and $(X_1^*, Y_1^*), \{(X^{(s)}, Y^{(s)})\}_{s \in S}$, obtaining ciphertexts $(D^*, E^*), \{(D^{(s)}, E^{(s)})\}_{s \in S}$. Note that we only need the D ciphertexts to be valid *left* inputs, and the E ciphertexts to be valid *right* inputs to `Comp`.
- For each $s \in S$, do the following:
  - Construct the matrix $V^{(s)}$, which consists of all the results of comparing d, e for $d \in D^{(s)}, e \in E^{(s)}$.
  - Compute the determinant polynomial on $V^{(s)}$, obtaining v_s.
- Find a linearly independent subset U of the v_s.
- Construct the matrix V^*, which consists of all the results of comparing d, e for $d \in D^*, e \in E^*$.
- Compute the determinant polynomial on V^*, obtaining v^*.
- Test if v^* is in the span of U. If it is, output 1, otherwise output 0.

From our prior analysis, $V^{(s)}$ is not full rank, so the determinant annihilates each of the $V^{(s)}$, giving a vector v_s in the idea $\langle hg \rangle$[5]. We will heuristically assume that the v_s span the entire ideal, which is plausible since the number of s, namely m', is much larger relative to the dimension of the ideal. Meanwhile, the determinant only annihilates V^* in the case $b = 1$. Thus v^* will be in $\langle hg \rangle$ if $b = 1$, but not if $b = 0$. Our linear independence test therefore distinguishes the two cases.

[5]This follows from the discussion in Section 4 and the fact that the determinant is homogeneous.

References

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

[BGK+14] Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Protecting obfuscation against algebraic attacks. In *Proc. of EuroCrypt*, 2014.

[BLR+15] Dan Boneh, Kevin Lewi, Mariana Raykova, Amit Sahai, Mark Zhandry, and Joe Zimmerman. Semantically secure order revealing encryption: Multi-input functional encryption without obfuscation. In *Proc. of EuroCrypt*, 2015.

[BMSZ15] Saikrishna Badrinarayanan, Eric Miles, Amit Sahai, and Mark Zhandry. Post-zeroizing obfuscation: The case of evasive circuits. Cryptology ePrint Archive, Report 2015/167, 2015. http://eprint.iacr.org/.

[BR14] Zvika Brakerski and Guy N. Rothblum. Virtual black-box obfuscation for all circuits via generic graded encoding. In *TCC*, 2014.

[CDPR15] Ronald Cramer, Léo Ducas, Chris Peikert, and Oded Regev. Recovering short generators of principal ideals in cyclotomic rings. *IACR Cryptology ePrint Archive*, 2015:313, 2015.

[CGH+15] Jean-Sébastien Coron, Craig Gentry, Shai Halevi, Tancrède Lepoint, Hemanta K. Maji, Eric Miles, Mariana Raykova, Amit Sahai, and Mehdi Tibouchi. Zeroizing without low-level zeroes: New MMAP attacks and their limitations. In *CRYPTO*, 2015.

[CJL16] Jung Hee Cheon, Jinhyuck Jeong, and Changmin Lee. An algorithm for cspr problems and cryptanalysis of the ggh multilinear map without an encoding of zero. Technical report, Cryptology ePrint Archive, Report 2016/139, 2016.

[GGH13a] S Garg, Craig Gentry, and S Halevi. Candidate multilinear maps from ideal lattices. *Advances in Cryptology (EuroCrypt)*, 2013.

[GGH+13b] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *Proc. of FOCS*, 2013.

[HJ15] Yupu Hu and Huiwen Jia. Cryptanalysis of GGH map. *IACR Cryptology ePrint Archive*, 2015:301, 2015.

[Kay09] Neeraj Kayal. The complexity of the annihilating polynomial. In *Proceedings of the 24th Annual IEEE Conference on Computational Complexity, CCC 2009, Paris, France, 15-18 July 2009*, pages 184–193, 2009.

[MSW14] Eric Miles, Amit Sahai, and Mor Weiss. Protecting obfuscation against arithmetic attacks. *IACR Cryptology ePrint Archive*, 2014:878, 2014.

[PST14] Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation from semantically-secure multilinear encodings. In *CRYPTO*, pages 500–517, 2014.