Cryptanalyses of Candidate Branching Program Obfuscators

Yilei Chen[1], Craig Gentry[2], and Shai Halevi[2]

[1] Boston University
chenyl@bu.edu
[2] IBM Research
craigbgentry@gmail.com, shaih@alum.mit.edu

Abstract. We describe new cryptanalytic attacks on the candidate branching program obfuscator proposed by Garg, Gentry, Halevi, Raykova, Sahai and Waters (GGHRSW) using the GGH13 graded encoding, and its variant using the GGH15 graded encoding as specified by Gentry, Gorbunov and Halevi. All our attacks require very specific structure of the branching programs being obfuscated, which in particular must have some input-partitioning property. Common to all our attacks are techniques to extract information about the "multiplicative bundling" scalars that are used in the GGHRSW construction.

For GGHRSW over GGH13, we show how to recover the ideal generating the plaintext space when the branching program has input partitioning. Combined with the information that we extract about the "multiplicative bundling" scalars, we get a distinguishing attack by an extension of the annihilation attack of Miles, Sahai and Zhandry. Alternatively, once we have the ideal we can solve the principle-ideal problem (PIP) in classical subexponential time or quantum polynomial time, hence obtaining a total break.

For the variant over GGH15, we show how to use the left-kernel technique of Coron, Lee, Lepoint and Tibouchi to recover ratios of the bundling scalars. Once we have the ratios of the scalar products, we can use factoring and PIP solvers (in classical subexponential time or quantum polynomial time) to find the scalars themselves, then run mixed-input attacks to break the obfuscation.

Keywords. Cryptanalysis, Graded-Encoding, Obfuscation.

1 Introduction

General-purpose code obfuscation is an amazingly powerful technique, making it possible to hide secrets in arbitrary running software. The first plausible construction of a secure general-purpose obfuscation, described three years ago by Garg, Gentry, Halevi, Raykova, Sahai and Waters [22] (hereafter GGHRSW), opened up a new direction of research that transformed our thinking about what can and cannot be done in cryptography. The GGHRSW construction consists of a "core component" for obfuscating branching programs, and a bootstrapping procedure that uses the core component — in conjunction with homomorphic encryption and some proofs — to obfuscate arbitrary code (modeled as a binary circuit). Many different constructions were proposed since then e.g., [12,8,37,3,27,34,7,39,6,23,11,4,31,32,24], most of which only modify the "core component" for branching programs, then use the GGHRSW bootstrapping to obfuscate circuits.

All known obfuscation constructions rely crucially on the underlying tool of *graded encoding schemes*, for which there are (essentially) only three candidate constructions: one due to Garg, Gentry and Halevi [21] (GGH13), another due to Coron, Lepoint and Tibouchi [19] (CLT13), and the third due to Gentry, Gorbunov and Halevi [26] (GGH15). However, the security properties of these encoding schemes are poorly understood, and therefore the same holds for the obfuscation constructions that use them.

Known attacks. The original publications of GGH13, CLT13 and GGH15 survey several number theoretical and algebraic attacks. For the GGH13 encoding scheme — that relies on the difficulty of the NTRU problem and the principle ideal problem (PIP) in certain number fields — we recently saw some advances in attacking these underlying problem [2,14,9,10,20], that may affect the choice of parameters.

The most serious attacks on all three encoding schemes are the so-called "zeroizing attacks": when encodings of zero are easy to find, some secrets can be extracted by linear algebraic techniques. The most devastating zeroizing attack is found by Cheon, Han, Lee, Ryu and Stehlé [13] against CLT13 — when the encodings of zero form certain combinations, one can extract all the secret parameters. The attack is extended by Coron et al. [16, Section 3.4], breaking the GGHRSW branching-program obfuscator when instantiated using CLT13 encodings and used to obfuscate branching programs with certain input-partitioning features.

Applying zeroizing attacks to construction based on GGH13 and GGH15 appears somewhat harder, especially in the context of obfuscation. Nonetheless, Miles, Sahai and Zhandry recently introduced "annihilation attack" against many GGH13-based branching-program obfuscators, for specific types of branching programs [35]. Interestingly, these attacks do not apply to the GGHRSW construction, due to the presence of some random entries in the encoded matrices. Moreover, it was shown in [24] that such random entries (in conjunction with other techniques) provably eliminates all known variants of zeroizing attacks.

To the best of our knowledge, no polynomial time attacks (either classical or quantum) were known before the current work on the GGHRSW obfuscator using GGH13 encoding, nor were there any attacks on any GGH15-based branching-program obfuscators.

This work. We describe new attacks on the GGHRSW branching-program obfuscator, when using GGH13 and GGH15 encodings. The attacks that we describe in this work require the underlying branching programs to satisfy some input-partitioning features, similar to the attack on the CLT variant [16, Section 3.4]. Roughly, the indexes of the branching program can be partitioned into two or three consecutive intervals, each contains "sufficiently many" input bits that do not appear in the other intervals.

A common thread in our attacks is that they focus on the "multiplicative bundling" scalars that are used in the GGHRSW construction (as protection against "mixed-input attacks"). We show that some information about these scalars can be extracted using zeroizing techniques, if the underlying branching program satisfy certain input-partitioning features. We are not able to fully recover these scalars, and hence cannot quite mount mixed-input attacks, but we can still use the extracted information in weaker attacks.

For the GGH13-based candidates, we first apply a variant of the attacks due to Cheon et al. and Coron et al. [13,17] to recover a basis of the ideal $\langle g \rangle$ that defines the plaintext space, as well as some representatives of the scalars, then use the recovered information in a distinguishing attack, using an extension of the annihilation attack of Miles et al. [35]. Alternatively, once we have a basis for $\langle g \rangle$ we can solve PIP (in classical subexponential time or quantum polynomial time), resulting in a total break.

For the GGH15-based candidates, we recover some rational expressions in the bundling scalars using techniques from [17] (among others), then we can use factoring and PIP solvers (in classical subexponential time or quantum polynomial time) to recover the bundling scalars themselves from the rational expressions, then mount mixed-input attacks.

Applicability and extensions of our attacks. We stress that all our attacks rely crucially on the input-partitioning of the branching program (in order to use the techniques of Cheon et al. or those of Coron et al.) In particular they do not seem to apply to "dual input" branching programs as used in many branching-program obfuscators. Also, our GGH13 attacks cannot be used against schemes that were proven secure in the "Weak Multilinear Map" model of Garg et al. [24], since our first step of recovering $\langle g \rangle$ fits in that model. However, some of our techniques do not seem to quite fit in that model (in particular Step II of the attack, see Section 3.2), so they should serve as a cautionary tale about relying too much on proofs of security in such idealized models. Also, the "immunization" techniques against GGH13 annihilation attack from [24] by themselves do not prevent our new attack if the branching programs are input-partitioning (see Section 3.5), it is only in combination with the "dual input" technique that they provide protection.

Finally, our techniques can potentially be combined with the recent techniques of Apon et al. and Coron et al. [5,18], to attack also some non-input-partitioned obfuscators. This seems a promising direction for future work.

2 Preliminaries

For a positive integer n, let $[n] = \{1, 2, ..., n\}$. Let Φ_n be the n^{th} cyclotomic polynomial. The typical ring used in the paper $R := \mathbb{Z}[x]/\langle\Phi_n(x)\rangle$, and the fractional field of R_n: $K_n := \mathbb{Q}[x]/\langle\Phi_n(x)\rangle$. Below we denote matrices by boldface uppercase letter (e.g., $\mathbf{A}, \mathbf{B}, \ldots$).

2.1 Matrix branching programs

We consider oblivious matrix branching programs (as usual in the obfuscation literature). Such a branching program consists of a sequence of steps, where each step is associated with an index of some input bit and we have two matrices associated with each step. To evaluate such a branching program over some input string, we choose one of the two matrices from each step, depending on the value of the corresponding input bit, then multiply all these matrices in order, and compare the result to the identity matrix.

Definition 1. *A dimension-w, length-h branching program over ℓ-bit inputs consists of an index-to-input map and a sequence of pairs of 0-1 matrices,*

$$\mathcal{B} = \left\{\iota : [h] \to [\ell],\ \{\mathbf{B}_{i,b} \in \{0,1\}^{w\times w}\}_{i\in[h],b\in\{0,1\}}\right\}.$$

This branching program is computing the function $f_{\mathcal{B}} : \{0,1\}^\ell \to \{0,1\}$, defined as

$$f_{\mathcal{B}}(x) = \begin{cases} 0 & \text{if } \prod_{i\in[h]} \mathbf{B}_{i,x_{\iota(i)}} = \mathbf{I} \\ 1 & \text{if } \prod_{i\in[h]} \mathbf{B}_{i,x_{\iota(i)}} \neq \mathbf{I} \end{cases}$$

where the matrix product is carried over some implicitly set ring that includes 0,1 (e.g., the ring R_n from above).

Input partitioning We say (somewhat informally) that a branching program $\mathcal{B}$ is input-partitioned if its set of steps can be partitioned into two or more consecutive intervals $[h] = \mathcal{H}_1||\mathcal{H}_2||\ldots$, such that for each interval there are "sufficiently many" input bits that control only steps in that interval and nowhere else. We sometime say that $\mathcal{B}$ is 2-partitioned or 3-partitioned if it can be broken to 2 or 3 intervals, respectively, and the number of bits that are unique to each interval will vary among the different attacks that we describe (and will typically be polylogarithmic).

When considering input-partitioned program $\mathcal{B}$, we will often consider its evaluation on inputs that differ in bits that only affect steps in one of the intervals. A simple (but important) observation that underlies most of our techniques is the following:

Lemma 1. *Let $\mathcal{B}$ be a branching program as per Definition 1 which is input-partitioned, $[h] = \mathcal{H}_1||\mathcal{H}_2$, and let $x, x' \in \{0,1\}^\ell$ be two zeros of $f_{\mathcal{B}}$ that differ only in bits that are mapped to steps in $\mathcal{H}_1$. Namely, $f_{\mathcal{B}}(x) = f_{\mathcal{B}}(x') = 0$,* *and for all $i \notin \mathcal{H}_1$ we have $x_{\iota(i)} = x'_{\iota(i)}$. Then the product of the matrices corresponding to $\mathcal{H}_1$ yields the same result in the evaluation of $\mathcal{B}$ on x and x', that is $\prod_{i\in\mathcal{H}_1} \mathbf{B}_{i,x_{\iota(i)}} = \prod_{i\in\mathcal{H}_1} \mathbf{B}_{i,x'_{\iota(i)}}$.*

Similarly, if x, x' are two zeros of $f_\mathcal{B}$ that differ only in bits that are mapped to steps in $\mathcal{H}_2$, then $\prod_{i\in\mathcal{H}_2} \mathbf{B}_{i,x_{\iota(i)}} = \prod_{i\in\mathcal{H}_2} \mathbf{B}_{i,x'_{\iota(i)}}$.

Proof. For the first statement, denote $\mathbf{B} := \prod_{i\in\mathcal{H}_1} \mathbf{B}_{i,x_{\iota(i)}}$, $\mathbf{B}' := \prod_{i\in\mathcal{H}_1} \mathbf{B}_{i,x'_{\iota(i)}}$, and $\mathbf{C} := \prod_{i\in\mathcal{H}_2} \mathbf{B}_{i,x_{\iota(i)}} = \prod_{i\in\mathcal{H}_2} \mathbf{B}_{i,x'_{\iota(i)}}$, where the last equality follows since $x_{\iota(i)} = x'_{\iota(i)}$ whenever $i \in \mathcal{H}_2$. Since $f_\mathcal{B}(x) = f_\mathcal{B}(x') = 0$ then $\mathbf{B}\times\mathbf{C} = \mathbf{B}'\times\mathbf{C} = \mathbf{I}$, and as $\mathbf{B}, \mathbf{B}', \mathbf{C}$ are square matrices then $\mathbf{C}$ must be invertible and $\mathbf{B} = \mathbf{B}' = \mathbf{C}^{-1}$. The proof of the "similarly" statement is analogous.

2.2 Overview of the GGHRSW branching-program obfuscator

We briefly review the candidate branching program obfuscator of Garg et al. [22] and its GGH15-based variant from [26, Section 5.2]. The GGHRSW branching-program obfuscator applies several different randomization steps to the underlying branching program, and then encodes the resulting randomized matrices, using either GGH13 or GGH15.

We defer the description of the GGH13 and GGH15 encoding schemes themselves to the corresponding attack sections, but just note that these schemes let us encode matrices in a way that allows checking whether certain degree-h polynomial expressions in these matrices evaluate to zero.

We also recall that these constructions are supposed to implement *indistinguishability obfuscation*. In the context of branching programs, this means that if two programs have the same length h and same input mapping function $\iota : [h] \to [\ell]$ and they compute the same function, then their obfuscations should be indistinguishable. Correspondingly when attacking these constructions we need to show two such equivalent programs for which we are able to distinguish the obfuscated versions.

Below we let $\mathcal{B} = \left\{\iota : [h] \to [\ell],\ \{\mathbf{B}_{i,b} \in \{0,1\}^{w\times w}\}_{i\in[h],b\in\{0,1\}}\right\}$ be the branching program to be obfuscated. The obfuscation process consists of the following steps:

0. Dummy branch. The construction begins by introducing a "dummy branch", which is just a length-h branching program with the same input mapping function $\iota : [h] \to [\ell]$, but consisting of only identity matrices of the same dimension as the $\mathbf{B}_{i,b}$'s. (In particular the "dummy branch" computes the all-zero function.) We refer to the original branching program as the "functional branch", and apply the same randomization/encoding transformations to both branches.

1. Random diagonal entries and bookends. Next every matrix in each of the branches (all are $w \times w$ 0-1 matrices) is embedded inside a higher-dimension randomized matrix. Specifically, for each $i \in [h], b \in \{0,1\}$ we consider the matrices

$$\tilde{\mathbf{B}}_{i,b} := \begin{bmatrix} \mathbf{V}_{i,b} & \\ & \mathbf{B}_{i,b} \end{bmatrix} \text{ and } \tilde{\mathbf{B}'}_{i,b} := \begin{bmatrix} \mathbf{V'}_{i,b} & \\ & \mathbf{I} \end{bmatrix}, \tag{1}$$

where $\mathbf{V}_{i,b}$ and $\mathbf{V'}_{i,b}$ are "random diagonal matrices." In the GGHRSW construction from [22], these are $2(h+3)$-by-$2(h+3)$ diagonal matrices with the diagonal entries chosen uniformly at random from the plaintext space, whereas in the GGH15-based variant from [26] they are diagonal 2-by-2 matrices with "random small entries" that are drawn from some Gaussian distribution over R_n. Below we denote the dimension of these random matrices as $2m$-by-$2m$ (so we have $m = h+3$ for the original GGHRSW and $m = 1$ for the GGH15-based variant). When the analysis requires fine grained structure of the padded matrices, we further split the notation for each m-by-m blocks and denote the whole as $\mathsf{diag}\left(\mathbf{U}_{i,b}, \mathbf{V}_{i,b}\right)$ and $\mathsf{diag}\left(\mathbf{U'}_{i,b}, \mathbf{V'}_{i,b}\right)$. The construction also chooses four "bookend" vectors $\mathbf{J}, \mathbf{J'}, \mathbf{L}, \mathbf{L'} \in R^{2m+w}$, of the form:

$$\mathbf{J}, \mathbf{J'} \in \left[0^m, \$^m, \$^w\right], \quad \mathbf{L}, \mathbf{L'} \in \left[\$^m, 0^m, \$^w\right]^T \tag{2}$$

where the \$'s stand for uniformly random elements from the plaintext space for the original GGH13-based construction, and for "small random" elements drawn from some Gaussian distribution for the GGH15-based candidate. They satisfy $\mathbf{JL} = \mathbf{J'L'}$.

2. **Killian-style randomization and bundling scalars.** Next the construction chooses invertible matrices $\{\mathbf{K}_i, \mathbf{K'}_i \in R_n^{(2m+w)\times(2m+w)}\}_{i\in[h]}$ and also scalars $\{\alpha_{i,b}, \alpha'_{i,b}\}_{i\in[h], b\in\{0,1\}}$. The scalars are chosen under the constraint that for any input bit $j \in [\ell]$, we have

$$\prod_{\iota(i)=j} \alpha_{i,0} = \prod_{\iota(i)=j} \alpha'_{i,0} \text{ and } \prod_{\iota(i)=j} \alpha_{i,1} = \prod_{\iota(i)=j} \alpha'_{i,1}.$$

Below we sometime use the notations $\beta_{j,b} := \prod_{\iota(i)=j} \alpha_{i,b}\left(= \prod_{\iota(i)=j} \alpha'_{i,b}\right)$. As before, here too the scalars and matrices are chosen at random from the plaintext space in the GGH13-based construction, and drawn from an appropriate Gaussian distribution with small parameters in the GGH15-based solution. Let us also denote $\mathbf{K}_0 = \mathbf{K'}_0 = \mathbf{I}$.

3. **Encoding.** Denote the randomized matrices by

$$\mathbf{S}_{i,b} := \alpha_{i,b}\mathbf{K}_{i-1}^{-1}\tilde{\mathbf{B}_{i,b}}\mathbf{K}_i \text{ and } \mathbf{S'}_{i,b} := \alpha'_{i,b}\mathbf{K'}_{i-1}^{-1}\tilde{\mathbf{B'}_{i,b}}\mathbf{K'}_i. \tag{3}$$

The obfuscation of the branching program $\mathcal{B}$ consists of encoding of all the matrices $\mathbf{S}_{i,b}$ and $\mathbf{S'}_{i,b}$ and also of the bookends $\mathbf{J}, \mathbf{J'}, \mathbf{L}, \mathbf{L'}$.

To evaluate the obfuscated branching program on some input x, we use the operations and zero-test capabilities of the underlying encoding scheme to check that $\mathbf{J}\left(\prod_{i\in[h]} \mathbf{S}_{i,b}\right)\mathbf{L} - \mathbf{J'}\left(\prod_{i\in[h]} \mathbf{S'}_{i,b}\right)\mathbf{L'} = 0$.

Branching program with input partitioning Let $\mathcal{X}||\mathcal{Y}||\mathcal{Z} = [h]$ be a 3-partition of the branching program steps. In the attacks we use honest evaluation of the branching program on many inputs of the form $u^{(i,j,k)} = x^{(i)}y^{(j)}z^{(k)}$, where all the bits that only affect steps in $\mathcal{X}$ are in the $x^{(i)}$ part, all the bits that only affect steps in $\mathcal{Y}$ are in the $y^{(j)}$ part, all the bits that only affect steps in $\mathcal{Z}$ are in the $z^{(k)}$ part, and all the other bits are fixed. This notation *does not mean* that the bits of $x^{(i)}$, $y^{(j)}$, $z^{(k)}$ appear in this order in $u^{(i,j,k)}$, but it does mean that $u^{(i,j,k)}$ and $u^{(i',j,k)}$ can only differ in bits that affect steps in $\mathcal{X}$, and similarly $u^{(i,j,k)}$ and $u^{(i,j',k)}$ only differ in bits that affect steps in $\mathcal{Y}$ and $u^{(i,j,k)}$ and $u^{(i,j,k')}$ only differ in bits that affect steps in $\mathcal{Z}$.

For such an input $u = xyz$, we denote by $\mathbf{S}_x$ the plaintext product matrix of functional branch in the $\mathcal{X}$ interval, by $\mathbf{S}_y$ the plaintext product matrix of functional branch in the the $\mathcal{Y}$ interval, and by $\mathbf{S}_z$ the plaintext product matrix of the functional branch in the $\mathcal{Z}$ interval (including the bookends). We similarly denote by $\mathbf{S}'_x$, $\mathbf{S}'_y$, $\mathbf{S}'_z$, the plaintext product matrix of the dummy branch. Namely

$$\begin{array}{l}\mathbf{S}_x := \mathbf{J} \cdot (\prod_{i\in\mathcal{X}} \mathbf{S}_{i,u_{\iota(i)}}),\ \ \mathbf{S}_y := \prod_{i\in\mathcal{Y}} \mathbf{S}_{i,u_{\iota(i)}},\ \mathbf{S}_z := (\prod_{i\in\mathcal{Z}} \mathbf{S}_{i,u_{\iota(i)}}) \cdot \mathbf{L}, \\ \mathbf{S}'_x := \mathbf{J}' \cdot (\prod_{i\in\mathcal{X}} \mathbf{S}'_{i,u_{\iota(i)}}),\ \mathbf{S}'_y := \prod_{i\in\mathcal{Y}} \mathbf{S}'_{i,u_{\iota(i)}},\ \mathbf{S}'_z := (\prod_{i\in\mathcal{Z}} \mathbf{S}'_{i,u_{\iota(i)}}) \cdot \mathbf{L}',\end{array} \tag{4}$$

with products over the plaintext space. In some cases we only need 2-partition of the program, so we suppress the $\mathbf{S}_y$, $\mathbf{S}'_y$ parts.

When we have multiple inputs of the form $u^{(i,j,k)} = x^{(i)}y^{(j)}z^{(k)}$ that are all zeros of the function, then by Lemma 1 the parts of the plaintext matrices that come from the product of the branching program matrices must be the same for the different $x^{(i)}$'s (and similarly for the different $y^{(j)}$'s and $z^{(k)}$'s). We denote these matrices simply by $\mathbf{B}_x$, $\mathbf{B}_y$, and $\mathbf{B}_z$, independently of i, j, k. Namely we have:

$$\begin{aligned}\mathbf{S}_{x^{(i)}} &= \alpha_{x^{(i)}}\ \mathbf{J}\ \times \mathsf{diag}(\mathbf{U}_{x^{(i)}}, \mathbf{V}_{x^{(i)}}, \mathbf{B}_x) \times \mathbf{K}_y; \\ \mathbf{S}'_{x^{(i)}} &= \alpha'_{x^{(i)}}\ \mathbf{J}'\ \times \mathsf{diag}(\mathbf{U}'_{x^{(i)}}, \mathbf{V}'_{x^{(i)}}, \mathbf{I}) \times \mathbf{K}'_y \\ \mathbf{S}_{y^{(j)}} &= \alpha_{y^{(j)}} \mathbf{K}_y^{-1} \times \mathsf{diag}(\mathbf{U}_{y^{(j)}}, \mathbf{V}_{y^{(j)}}, \mathbf{B}_y) \times \mathbf{K}_z; \\ \mathbf{S}'_{y^{(j)}} &= \alpha'_{y^{(j)}} \mathbf{K}'^{-1}_y \times \mathsf{diag}(\mathbf{U}'_{y^{(j)}}, \mathbf{V}'_{y^{(j)}}, \mathbf{I}) \times \mathbf{K}'_z; \\ \mathbf{S}_{z^{(k)}} &= \alpha_{z^{(k)}} \mathbf{K}_z^{-1} \times \mathsf{diag}(\mathbf{U}_{z^{(k)}}, \mathbf{V}_{z^{(k)}}, \mathbf{B}_z) \times \mathbf{L}; \\ \mathbf{S}'_{z^{(k)}} &= \alpha'_{z^{(k)}} \mathbf{K}'^{-1}_z \times \mathsf{diag}(\mathbf{U}'_{z^{(k)}}, \mathbf{V}'_{z^{(k)}}, \mathbf{I}) \times \mathbf{L}'\end{aligned} \tag{5}$$

where the scalars $\alpha_{x^{(i)}}$, $\alpha_{y^{(j)}}$, etc. are just the product of all the $\alpha_{i,b}$'s in the corresponding (partial) branch. Moreover, we observe that all the ratios of $\alpha_{x^{(i)}}/\alpha'_{x^{(i)}}$, $i = 1, 2, \ldots$ (and similarly for the $\alpha_{y^{(j)}}$ and $\alpha_{z^{(k)}}$) must also be equal.

Lemma 2. *With the notations above, we have* $\alpha'_{x^{(1)}}/\alpha_{x^{(1)}} = \alpha'_{x^{(2)}}/\alpha_{x^{(2)}} = \ldots$ *and similarly* $\alpha'_{y^{(1)}}/\alpha_{y^{(1)}} = \alpha'_{y^{(2)}}/\alpha_{y^{(2)}} = \ldots$ *and* $\alpha'_{z^{(1)}}/\alpha_{z^{(1)}} = \alpha'_{z^{(2)}}/\alpha_{z^{(2)}} = \ldots$*.*

Proof. To prove the statement for the $\alpha_{x^{(i)}}$'s consider an input bit $t \in [\ell]$ that affect some steps in $\mathcal{X}$. That bit either only affects steps in $\mathcal{X}$ or it affects steps in both $\mathcal{X}$ and in $\mathcal{Y}, \mathcal{Z}$. In the former case, by construction we have $\prod_{\iota(i')=t} \alpha_{i',b} = \prod_{\iota(i')=t} \alpha'_{i',b}$ (for $b = 0, 1$), so this input bit's contribution to the ratio $\alpha'_{x^{(i)}}/\alpha_{x^{(i)}}$ is 1 (for all i). In the latter case, this input bit has the same value (0 or 1) for all the inputs $x^{(i)}$, so it contributes the same factor to the ratio $\alpha'_{x^{(i)}}/\alpha_{x^{(i)}}$ for all i. The proof for the $\alpha_{y^{(j)}}$ and $\alpha_{z^{(k)}}$ is the same.

3 Cryptanalysis of the GGH13-based candidate

The GGH13 Encoding Scheme. The core secret parameter in the GGH13 encoding scheme is a small $g \in R_n$ (sampled from small Gaussian distribution), such that the inverse $g^{-1} \in K$ is also small. Let $\mathcal{I} = \langle g \rangle = gR_n$ be the ideal generated by g in R_n, the plaintext space of the GGH13 scheme is the quotient ring $R_n/\mathcal{I}$, and we typically choose g so that this plaintext space is isomorphic to some prime field $\mathbb{F}_p$. Other parameters of the scheme are an integer modulus $q \gg p$ and the multi-linearity degree k (which are public), and a random secret denominator $z \in R_n/qR_n$ (which is kept secret). Plaintext elements are encoded relative to levels between 0 and k.

The encoding of $s \in R_n/\mathcal{I}$ at level 0 is a short representative of the coset of the ideal shifted by s, i.e. $c \in s + \mathcal{I}$, $\|c\| \ll q$. To encode at level i, compute $c/z^i \pmod q$. (There is also an "asymmetric mode" of GGH13, in which there are many different denominators z_i.) The public zero-test parameter is $p_{zt} = \eta \cdot z^k/g$, with $\|\eta\| \leq q^{1/2}$. [3] Additions and multiplications are simply adding and multiplying the encodings in R_n/qR_n, with the restrictions that correctness only holds when adding on the same level, or multiplying below the maximum level k. To zero-test, multiply the (potential) top-level encoding c/z^k by p_{zt} (modulo q). If c encodes zero then $c \in \mathcal{I}$, hence $c = c' \cdot g$, and therefore $c \cdot p_{zt} = \eta c'$, which is small since both η and c' are much smaller than q.

Attacking the GGH13-based obfuscator When using GGH13 as the underlying encoding scheme in the GGHRSW obfuscator, we denote the encoding of the plaintext matrices $\mathbf{S}_{i,b}$, $\mathbf{S}'_{i,b}$ by

$$\mathbf{C}_{i,b} = (\mathbf{S}_{i,b} + g \cdot \mathbf{E}_{i,b})/z, \text{ and } \mathbf{C}'_{i,b} = (\mathbf{S}'_{i,b} + g \cdot \mathbf{E}'_{i,b})/z.$$

We also denote the encoding of the bookends by

$$\tilde{\mathbf{J}} = (\mathbf{J}+g\cdot\mathbf{E}_J)/z,\ \tilde{\mathbf{L}} = (\mathbf{L}+g\cdot\mathbf{E}_L)/z,\ \tilde{\mathbf{J}}' = (\mathbf{J}'+g\cdot\mathbf{E}'_{J'})/z, \text{ and } \tilde{\mathbf{L}}' = (\mathbf{L}'+g\cdot\mathbf{E}'_{L'})/z,$$

where all the calculations are modulo q.

We first recover the ideal $\langle g \rangle$ adapting the zeroing attack techniques of Cheon, Han, Lee, Ryu and Stehlé [13] and Coron, Lee, Lepoint and Tibouchi [17]. This part requires 2-partitioning of the branching program. Once we have a basis of $\langle g \rangle$, sub-exponential time classical algorithms [9] and polynomial-time quantum algorithms [10] are known to recover a short generator of $\langle g \rangle$ [20], thus breaking GGH13 completely [21, Section 6.3.3].

Alternatively, using a basis of $\langle g \rangle$ we can proceed with the zeroing attack modulo $\langle g \rangle$ to recover (some representation of) products of the bundling scalars.

[3] The scalar η is denoted h in [21], but we are already using h for the length of the branching program.

Then we can execute a simplified variant of the annihilation attack by Miles, Sahai and Zhandry [35]. This yields a classical polynomial time attack, and requires 3-partitioning of the branching program. We now proceed to describe the attack in more details.

Some more notations. Consider a 3-partitioned branching program with the partitioning $\mathcal{X}||\mathcal{Y}||\mathcal{Z} = [h]$. We use the same notation as in Eqn. (4) for the plaintext matrices, and also denote $\mathbf{C}_x, \mathbf{C}_y, \mathbf{C}_z$ and $\mathbf{C}'_x, \mathbf{C}'_y, \mathbf{C}'_z$ for the encoded matrices. Namely

$$\begin{aligned}\mathbf{C}_x &:= \tilde{\mathbf{J}} \cdot (\textstyle\prod_{i\in\mathcal{X}} \mathbf{C}_{i,u_{\iota(i)}}),\ \mathbf{C}_y := \textstyle\prod_{i\in\mathcal{Y}} \mathbf{C}_{i,u_{\iota(i)}},\ \mathbf{C}_z := (\textstyle\prod_{i\in\mathcal{Z}} \mathbf{C}_{i,u_{\iota(i)}}) \cdot \tilde{\mathbf{L}},\\ \mathbf{C}'_x &:= \tilde{\mathbf{J}'} \cdot (\textstyle\prod_{i\in\mathcal{X}} \mathbf{C}'_{i,u_{\iota(i)}}), \mathbf{C}'_y := \textstyle\prod_{i\in\mathcal{Y}} \mathbf{C}'_{i,u_{\iota(i)}},\ \mathbf{C}'_z := (\textstyle\prod_{i\in\mathcal{Z}} \mathbf{C}'_{i,u_{\iota(i)}}) \cdot \tilde{\mathbf{L}'},\end{aligned}$$

with products over R_n/qR_n. As before, when we only need 2-partition we ignore the $\mathbf{C}_y$'s. With these notations, for any $u = xyz$ we can multiply, subtract, and zero-test to get

$$\begin{aligned} w &:= p_{zt}(\mathbf{C}_x\mathbf{C}_y\mathbf{C}_z - \mathbf{C}'_x\mathbf{C}'_y\mathbf{C}'_z) \qquad (6)\\ &= \frac{\eta}{g} \cdot [\mathbf{S}_x + g\mathbf{E}_x, -(\mathbf{S}'_x + g\mathbf{E}'_x)] \begin{bmatrix}\mathbf{S}_y + g\mathbf{E}_y, & 0\\ 0, & \mathbf{S}'_y + g\mathbf{E}'_y\end{bmatrix} \begin{bmatrix}\mathbf{S}_z + g\mathbf{E}_z\\ \mathbf{S}'_z + g\mathbf{E}'_z\end{bmatrix} \pmod q \end{aligned}$$

(or the without the middle matrix if we only use 2-partitioning). Moreover, if u is a zero of the function then the final zero-tested value is an encoding of zero, and hence Eqn. (6) holds not only modulo q but also over the base ring R_n.

3.1 Step I: recovering $\langle g\rangle$

Our first task is to recover (a basis for) the plaintext-space ideal $\mathcal{I} = \langle g\rangle$. To that end, we will construct two matrices $\mathbf{M}, \mathbf{N}$ which are both full rank over R_n (whp), but (after canceling some common factors) the determinant of $\mathbf{M}$ is divisible by a higher power of g than the determinant of $\mathbf{N}$. Computing $\mathbf{M} \times \mathbf{N}^{-1}$ over the field of fractions K_n and multiplying by the common denominator, we get an integral matrix whose determinant is divisible by g. Repeating this process many times and taking the common denominator of all the resulting determinants we obtain whp a basis for the ideal $\langle g\rangle$.

Let $\mathcal{X}||\mathcal{Z} = [h]$ be a 2-partition of the branching program steps, where we have sufficiently many input bits that only affect steps in the $\mathcal{X}$ interval and sufficiently many other input bits that only affect steps in the $\mathcal{Z}$ interval. (Denote these input bits by $J_x, J_z \subset [\ell]$, respectively.) Moreover, we can fix all the remaining input bits in such a way that for sufficiently many choices $x^{(i)} \in \{0,1\}^{|J_x|}$, $z^{(j)} \in \{0,1\}^{|J_z|}$ we get an input which is a zero of the function.

Finally, we assume that there are two distinguished input bits $j_1, j_2 \in J_x$ that we can set arbitrarily. Namely, for all the other choices of input bits as above, we can set these two bits to 00,01,10, and 11 and all four combinations will yield a zero of the function.

With these assumptions, let us denote by $w_{00}^{(i,j)}$ the zero-tested value which was obtained by honest evaluation of the obfuscated program on the input $x_{00}^{(i)}z^{(j)}$ with the two distinguished bits set to 00, and similarly $w_{01}^{(i,j)}$, $w_{10}^{(i,j)}$, $w_{11}^{(i,j)}$ with these bits set to 01, 10, 11, respectively. Note that:

- For every fixed i, j, the four inputs whose evaluation yields the scalars $w_{00}^{(i,j)}$, $w_{01}^{(i,j)}$, $w_{10}^{(i,j)}$, and $w_{11}^{(i,j)}$ differ only in the values of the distinguished input bit;
- For every $a \in \{00, 01, 10, 11\}$ and every fixed j, the inputs whose evaluation yields the different $\{w_a^{(i,j)}\}_i$ only differ in bits that affect the $\mathcal{X}$ interval of steps (but not the distinguished j_1, j_2); and
- For every $a \in \{00, 01, 10, 11\}$ and every fixed i, the inputs whose evaluation yields the different $\{w_a^{(i,j)}\}_j$ only differ in bits that affect the $\mathcal{Z}$ interval of steps.

Using Eqn. (6), we have for all i, j and $a \in \{00, 01, 10, 11\}$,

$$\begin{aligned} w_a^{(i,j)} &:= p_{zt}\big(\mathbf{C}_{x_a^{(i)}}\mathbf{C}_{z^{(j)}} - \mathbf{C}'_{x_a^{(i)}}\mathbf{C}'_{z^{(j)}}\big) \\ &= \frac{\eta}{g} \cdot \left[(\mathbf{S}_{x_a^{(i)}} + g\mathbf{E}_{x_a^{(i)}})(\mathbf{S}_{z^{(j)}} + g\mathbf{E}_{z^{(j)}}) - (\mathbf{S}'_{x_a^{(i)}} + g\mathbf{E}'_{x_a^{(i)}})(\mathbf{S}'_{z^{(j)}} + g\mathbf{E}'_{z^{(j)}})\right] \\ &= \frac{\eta}{g} \cdot \left[\mathbf{S}_{x_a^{(i)}} + g\mathbf{E}_{x_a^{(i)}},\ (-\mathbf{S}'_{x_a^{(i)}} - g\mathbf{E}'_{x_a^{(i)}})\right] \begin{bmatrix}\mathbf{S}_{z^{(j)}} + g\mathbf{E}_{z^{(j)}} \\ \mathbf{S}'_{z^{(j)}} + g\mathbf{E}'_{z^{(j)}}\end{bmatrix} \end{aligned} \tag{7}$$

with Eqn. (7) holding over the base ring R_n. Fixing $a \in \{00, 01, 10, 11\}$ and letting i, j range over sufficiently many inputs, we get the matrices

$$\begin{aligned} \mathbf{W}_a &:= [w_a^{(i,j)}]_{i,j} = \mathbf{X}_a\mathbf{Z} \\ &:= \frac{\eta}{g}\begin{bmatrix} \cdots \\ \mathbf{S}_{x_a^{(i)}} + g\mathbf{E}_{x_a^{(i)}}, (-\mathbf{S}'_{x_a^{(i)}} - g\mathbf{E}'_{x_a^{(i)}}) \\ \cdots \end{bmatrix} \begin{bmatrix} \cdots, \mathbf{S}_{z^{(j)}} + g\mathbf{E}_{z^{(j)}}, \cdots \\ \cdots, \mathbf{S}'_{z^{(j)}} + g\mathbf{E}'_{z^{(j)}}, \cdots \end{bmatrix} \end{aligned} \tag{8}$$

Specifically we choose as many different $x^{(i)}$'s and $z^{(j)}$'s to make $\mathbf{X}_a$ and $\mathbf{Z}$ square matrices (of dimension 2ρ, where $\rho = 2m + w$).

The two matrices $\mathbf{M}, \mathbf{N}$ that we consider in this part of the attack are

$$\begin{aligned} \mathbf{M} = \begin{bmatrix}\mathbf{W}_{00} & \mathbf{W}_{01} \\ \mathbf{W}_{10} & \mathbf{W}_{11}\end{bmatrix} &= \tfrac{\eta}{g} \cdot \begin{bmatrix}\mathbf{X}_{00} & \mathbf{X}_{01} \\ \mathbf{X}_{10} & \mathbf{X}_{11}\end{bmatrix} \times \begin{bmatrix}\mathbf{Z} & \\ & \mathbf{Z}\end{bmatrix}, \\ \mathbf{N} = \begin{bmatrix}\mathbf{W}_{00} & 0 \\ 0 & \mathbf{W}_{11}\end{bmatrix} &= \tfrac{\eta}{g} \cdot \begin{bmatrix}\mathbf{X}_{00} & 0 \\ 0 & \mathbf{X}_{11}\end{bmatrix} \times \begin{bmatrix}\mathbf{Z} & \\ & \mathbf{Z}\end{bmatrix} \end{aligned} \tag{9}$$

These matrices will have full rank over the base ring R_n whp due to the "random" error matrices $\mathbf{E}$ in the $\mathbf{X}$'s and $\mathbf{Z}$. However, we show now that whp, the determinant of $\mathbf{M}$ (after disregarding the common factor $\frac{\eta}{g}$) is divisible by a higher power of g than that of $\mathbf{N}$.

To see that, recall that the matrices $\mathbf{S}_x$ from Eqn. (8) are the plaintext matrices of the GGHRSW constructions as per Eqn. (5), and in particular they include the scalars $\beta_{j_1,b}$, $\beta_{j_2,b}$ for the two distinguished input bits j_1, j_2. To somewhat simplify notations we use below $\beta_b := \beta_{j_1,b}$ and $\beta'_b = \beta_{j_2,b}$. Specifically for any index i we have

$$\begin{aligned}
\mathbf{S}_{x^{(i)}_{00}} = &\quad \beta_0\beta'_0 \cdot \gamma^{(i)} \cdot \ \mathbf{J} \ \times \mathsf{diag}(\mathbf{U}^{(i)}_{00}, \mathbf{V}^{(i)}_{00}, \mathbf{B}_x) \times \mathbf{K}_z \\
&= \ \beta_0\beta'_0 \cdot \gamma^{(i)} \cdot \left[0,\ \mathbf{v}^{(i)}_{00},\ \mathbf{b}\right] \times \mathbf{K}_z \pmod{\mathcal{I}} \\
\mathbf{S'}_{x^{(i)}_{00}} = &\quad \delta \cdot \beta_0\beta'_0 \cdot \gamma^{(i)} \cdot \ \mathbf{J'} \ \times \mathsf{diag}(\mathbf{U'}^{(i)}_{00}, \mathbf{V'}^{(i)}_{00}, \mathbf{I}) \times \mathbf{K'}_z \\
&= \ \delta \cdot \beta_0\beta'_0 \cdot \gamma^{(i)} \cdot \left[0,\ \mathbf{v'}^{(i)}_{00},\ \mathbf{b'}\right] \times \mathbf{K'}_z \pmod{\mathcal{I}},
\end{aligned} \tag{10}$$

where above we used $\delta := \alpha_{x^{(i)}_{\sigma\tau}} / \alpha'_{x^{(i)}_{\sigma\tau}}$ (which by Lemma 2 is independent of i or the two bits σ, τ), and $\gamma^{(i)}$ is some scalar that depends on i but not on these two bits. We use similar notations for $\mathbf{S}_{x^{(i)}_{01}}$, $\mathbf{S}_{x^{(i)}_{10}}$, $\mathbf{S}_{x^{(i)}_{11}}$. For any two bits σ, τ, each row i of $\mathbf{X}_{\sigma\tau}$ (mod $\mathcal{I}$) has the form $[\mathbf{S}_{x^{(i)}_{\sigma\tau}} | - \mathbf{S'}_{x^{(i)}_{\sigma\tau}}]$, so we can write

$$\mathbf{X}_{\sigma\tau} = \left(\beta_\sigma\beta'_\tau \cdot \mathbf{X} + \boldsymbol{\Delta}_{\sigma\tau}\right) \times \mathsf{diag}(\mathbf{K}_z, \mathbf{K'}_z) \pmod{\mathcal{I}} \tag{11}$$

Where $\mathsf{diag}(\mathbf{K}_z, \mathbf{K'}_z)$ is invertible, $\mathbf{X}$ is some fixed matrix independent of σ, τ, and where $\Delta_{\sigma\tau}$ has only few non-zero columns (i.e., the ones corresponding to $\mathbf{v}^{(i)}_{\sigma\tau}$ and $\mathbf{v'}^{(i)}_{\sigma\tau}$ from Eqn. (10)). Denoting by n the number of non-zero columns in the Δ's, we have (over $R_n/\mathcal{I}$)

$$\begin{aligned}
\mathsf{rank}\begin{pmatrix} \beta_0\beta'_0\mathbf{X} + \Delta_{00} & \beta_0\beta'_1\mathbf{X} + \Delta_{01} \\ \beta_1\beta'_0\mathbf{X} + \Delta_{10} & \beta_1\beta'_1\mathbf{X} + \Delta_{11} \end{pmatrix} &\leq 2n + \mathsf{rank}\begin{pmatrix} \beta_0\beta'_0\mathbf{X} & \beta_0\beta'_1\mathbf{X} \\ \beta_1\beta'_0\mathbf{X} & \beta_1\beta'_1\mathbf{X} \end{pmatrix} \\
&= 2n + \mathsf{rank}(\mathbf{X}),
\end{aligned}$$

because $\beta_0\beta'_0 \cdot \beta_1\beta'_1 - \beta_0\beta'_1 \cdot \beta_1\beta'_0 = 0$. On the other hand, we have

$$\mathsf{rank}\begin{pmatrix} \beta_0\beta'_0\mathbf{X} + \Delta_{00} & 0 \\ 0 & \beta_1\beta'_1\mathbf{X} + \Delta_{11} \end{pmatrix} \overset{(whp)}{=} 2n + 2 \cdot \mathsf{rank}(\mathbf{X}).$$

Since it has lower rank modulo $\mathcal{I}$, then (at least heuristically[4]) the determinant of $\begin{bmatrix} \mathbf{X}_{00} & \mathbf{X}_{01} \\ \mathbf{X}_{10} & \mathbf{X}_{11} \end{bmatrix}$ is divisible by a higher power of g than that of $\begin{bmatrix} \mathbf{X}_{00} & 0 \\ 0 & \mathbf{X}_{11} \end{bmatrix}$.

Computing $\mathbf{MN}^{-1}$ over K, the common factor η/g drops out, and we are left with a fractional matrix such that

$$\det(\mathbf{MN}^{-1}) = \det\begin{pmatrix} \mathbf{X}_{00} & \mathbf{X}_{01} \\ \mathbf{X}_{10} & \mathbf{X}_{11} \end{pmatrix} / \det\begin{pmatrix} \mathbf{X}_{00} & 0 \\ 0 & \mathbf{X}_{11} \end{pmatrix} \ = \ \frac{\text{a multiple of } g}{\text{some denominator}} \ ,$$

[4] Having $\mathsf{rank}(\mathbf{A}) > \mathsf{rank}(\mathbf{B}) \pmod{g}$ does not always mean that $\det(\mathbf{B})$ is divisible by a higher power of g than $\det(\mathbf{A})$, since $\mathbf{A}$ could have one eigenvalue which is divisible by a high power of g, e.g., consider $\mathbf{A} = \begin{bmatrix} g^5 & 0 \\ 0 & 1 \end{bmatrix}$ and $\mathbf{B} = \begin{bmatrix} g & 0 \\ 0 & g \end{bmatrix}$. For our "random matrices", however, this is unlikely, as confirmed by our experiments.

where the denominator is not divisible by g. Multiplying by the denominator we thus get a multiple of g, as needed. Repeating this process several times with different distinguished indexes j_1, j_2, we can take the GCD and whp get a basis for some power $\mathcal{I}^t$ of the ideal $\mathcal{I}$.

Finally, when $\mathcal{I}$ is a prime ideal then it is easy to find $\mathcal{I}$ from $\mathcal{I}^t$: The norm of $\mathcal{I}^t$ is $\mathsf{norm}(\mathcal{I})^r$, and $p = \mathsf{norm}(\mathcal{I})$ is a prime integer, and we can find p from p^t (by exhaustive search over t). The Kummer-Dedekind theorem let us compute all the ideals of norm p in K, and one of these ideals is $\mathcal{I}$.

3.2 Step II: recovering some representatives of the bundling scalars

For this step we need the branching program to be 3-partitioned. Recall that Eqn. (6) holds over R if the input $u = x^{(i)}y^{(b)}z^{(j)}$ is a zero of the function. Let i, j ranging over 2ρ inputs, and for $b \in \{0, 1\}$, we get the matrices:

$$\begin{aligned}
\mathbf{W}_b :=\; & \mathbf{X}\mathbf{Y}_b\mathbf{Z} \\
:=\; & \frac{\eta}{g}\begin{bmatrix} \cdots \\ \mathbf{S}_{x^{(i)}} + g\mathbf{E}_{x^{(i)}}, -(\mathbf{S}'_{x^{(i)}} + g\mathbf{E}'_{x^{(i)}}) \\ \cdots \end{bmatrix} \\
& \cdot \begin{bmatrix} \mathbf{S}_{y^{(b)}} + g\mathbf{E}_{y^{(b)}}, & 0 \\ 0, & \mathbf{S}'_{y^{(b)}} + g\mathbf{E}'_{y^{(b)}} \end{bmatrix} \cdot \begin{bmatrix} \cdots, \mathbf{S}_{z^{(j)}} + g\mathbf{E}_{z^{(j)}}, \cdots \\ \cdots, \mathbf{S}'_{z^{(j)}} + g\mathbf{E}'_{z^{(j)}}, \cdots \end{bmatrix}
\end{aligned} \tag{12}$$

where $\mathbf{X}$, $\mathbf{Y}_1$, $\mathbf{Y}_0$, $\mathbf{Z} \in R^{2\rho\times 2\rho}$ are full-rank w.h.p. due to the contribution of $\mathbf{E}$ terms from different paths.

We then compute the characteristic polynomial χ of $\mathbf{W}_1\mathbf{W}_0^{-1} \in K^{2\rho\times 2\rho}$, which is equal to the characteristic polynomial of $\mathbf{Y}_1\mathbf{Y}_0^{-1}$. Considering $\mathbf{Y}_1\mathbf{Y}_0^{-1}$ modulo $\mathcal{I}$ we have:

$$\begin{aligned}
\mathbf{Y}_1\mathbf{Y}_0^{-1} &= \begin{bmatrix} \mathbf{S}_{y^{(1)}} + g\mathbf{E}_{y^{(1)}}, & 0 \\ 0, & {\mathbf{S}'}_{y^{(1)}} + g{\mathbf{E}'}_{y^{(1)}} \end{bmatrix} \begin{bmatrix} \mathbf{S}_{y^{(0)}} + g\mathbf{E}_{y^{(0)}}, & 0 \\ 0, & {\mathbf{S}'}_{y^{(0)}} + g{\mathbf{E}'}_{y^{(0)}} \end{bmatrix}^{-1} \\
&= \begin{bmatrix} \mathbf{S}_{y^{(1)}}, & 0 \\ 0, & {\mathbf{S}'}_{y^{(1)}} \end{bmatrix} \begin{bmatrix} \mathbf{S}_{y^{(0)}}, & 0 \\ 0, & {\mathbf{S}'}_{y^{(0)}} \end{bmatrix}^{-1} \pmod{\mathcal{I}}
\end{aligned} \tag{13}$$

Expanding the "functional term" of $\mathbf{Y}_1\mathbf{Y}_0^{-1} \pmod{\mathcal{I}}$, i.e. $\mathbf{S}_{y^{(1)}}\mathbf{S}_{y^{(0)}}^{-1}$, we have:

$$\begin{aligned}
\mathbf{S}_{y^{(1)}}\mathbf{S}_{y^{(0)}}^{-1} =\; & \alpha_{y^{(1)}}\mathbf{K}_x^{-1} \begin{bmatrix} \mathbf{U}_{y^{(1)}}, & 0, & 0 \\ 0, & \mathbf{V}_{y^{(1)}}, & 0 \\ 0, & 0, & \mathbf{B}_{y^{(1)}} \end{bmatrix} \mathbf{K}_z \\
& \times \left(\alpha_{y^{(0)}}\mathbf{K}_x^{-1} \begin{bmatrix} \mathbf{U}_{y^{(0)}}, & 0, & 0 \\ 0, & \mathbf{V}_{y^{(0)}}, & 0 \\ 0, & 0, & \mathbf{B}_{y^{(0)}} \end{bmatrix} \mathbf{K}_z \right)^{-1} \\
=\; & \frac{\alpha_{y^{(1)}}}{\alpha_{y^{(0)}}} \cdot \mathbf{K}_x^{-1} \begin{bmatrix} \mathbf{U}_{y^{(1)}}\mathbf{U}_{y^{(0)}}^{-1}, & 0, & 0 \\ 0, & \mathbf{V}_{y^{(1)}}\mathbf{V}_{y^{(0)}}^{-1}, & 0, \\ 0, & 0, & \mathbf{B}_{y^{(1)}}\mathbf{B}_{y^{(0)}}^{-1} \end{bmatrix} \mathbf{K}_x
\end{aligned} \tag{14}$$

By Lemma 1, $\mathbf{B}_{y^{(1)}}\mathbf{B}_{y^{(0)}}^{-1} = \mathbf{I}^{w\times w}$, so $\alpha_{y^{(1)}}/\alpha_{y^{(0)}} \in K$ is an eigenvalue of $\mathbf{S}_{y^{(1)}}\mathbf{S}_{y^{(0)}}^{-1}$ with multiplicity at least the dimensions of the $\mathbf{B}$'s (i.e., at least w).[5] Similarly $\alpha'_{y^{(1)}}/\alpha'_{y^{(0)}}$ is an eigenvalue of $\mathbf{S}'_{y^{(1)}}\mathbf{S}'^{-1}_{y^{(0)}}$ of multiplicity at least w, and by Lemma 2 we have $\alpha'_{y^{(1)}}/\alpha'_{y^{(0)}} = \alpha_{y^{(1)}}/\alpha_{y^{(0)}}$. Hence $\alpha_{y^{(1)}}\alpha_{y^{(0)}}^{-1}$ is the eigenvalue of $\mathbf{Y}_1\mathbf{Y}_0^{-1} \pmod{\mathcal{I}}$ of multiplicity at least $2w$. Given a basis of $\mathcal{I}$, we can solve the characteristic polynomial $\chi_{\mathbf{W}_1\mathbf{W}_0^{-1}} \pmod{\mathcal{I}}$ and obtain eigenvalues in K. The eigenvalue of multiplicity $2w$ is $\alpha_{y^{(1)}}\alpha_{y^{(0)}}^{-1}$.

3.3 Step III: Annihilation Attack

The annihilation attacks described by Miles, Sahai and Zhandry [35] do not extend to break GGH13-based branching program obfuscators with the padded random diagonal entries. We show that with the knowledge of the ratios of scalars (even if their representations are big), this attack can be extended to handle the random diagonal entries. We begin with a brief overview of the attacks from [35].

Given many level-0 encodings $\{c_i = s_i + e_i \cdot g\}_i$, any degree-$d$ expression in them can be written as

$$c = r_0 + r_1 \cdot g^1 + r_2 \cdot g^2 + \ldots + r_d \cdot g^d \pmod{q}.$$

If that expression is encoded at level d, then multiplying it by the zero-test parameter yields $x = p_{zt} \cdot c/z^d = h(r_0 g^{-1} + r_1 + r_2 g + \ldots r_d g^{d-1}) \pmod{q}$ (which is small if $r_0 = 0$ and likely large when $r_0 \neq 0$).

An annihilation attack consists of collecting and zero-testing many encodings with $r_0 = 0$, getting the corresponding $x^{(1)}, x^{(2)}, \ldots$, then applying some carefully-selected polynomial to these $x^{(i)}$'es and examining the result. Specifically, Miles et al. observed that it is possible to check whether or not the terms that depends only on the r_1 values vanish in the resulting polynomial. They also observed that these r_1 values can be expressed as very structured expressions in the encoded secret and the error terms,

$$r_1 = e_1 s_2 \ldots s_d + s_1 e_2 s_3 \ldots s_d + \ldots + s_1 s_2 \ldots e_d.$$

Using these observation, Miles et al. described in [35] a particular polynomial in the $x^{(i)}$'s that can be used to distinguish the obfuscation of equivalent branching programs (under some contemporary obfuscators).

Introducing our running example. To help describe our attack, we show below how it can be used to distinguish between GGHRSW obfuscation of two specific branching programs that compute the constant zero function. For this attack we need the branching programs to be 3-partitioned with intervals $\mathcal{X}||\mathcal{Y}||\mathcal{Z} = [h]$, and we need to have two distinguished input bit positions j_1, j_2 that only control steps in the $\mathcal{Y}$ interval but not $\mathcal{X}$ or $\mathcal{Z}$. In addition, we require that bit j_1 controls

[5] We remark that this step of finding (the multiplicity of) an eigenvalue does not seem to fit in the "Weak Multilinear Map" model of Garg et al. [24].

at least two steps (denoted u, w) in the $\mathcal{Y}$ interval, and that bit j_2 controls (at least) one step in between u and w (denoted v). That is, we need $u, v, w \in \mathcal{Y}$ with $u < v < w$, such that $\iota(u) = \iota(w) = j_1$, $\iota(v) = j_2$, and j_j does not control any steps before u or after w. As before, we shorten our notations somewhat and denote the relevant products of the bundling constants by

$$\beta_0 := \prod_{\iota(i)=j_1} \alpha_{i,0}, \quad \beta_1 := \prod_{\iota(i)=j_1} \alpha_{i,1}, \quad \beta_0' := \prod_{\iota(i)=j_2} \alpha_{i,0}, \quad \beta_1' := \prod_{\iota(i)=j_2} \alpha_{i,1}.$$

The two branching programs in our running example will have the identity matrix for both 0 and 1 in all the steps *except for the two steps u, w controlled by y_1*, and the zero matrices will be the identity also for these two steps. For the 1 matrices in these two steps, in one program they too will be the identity, and in the other program those two matrices are a permutation matrix and its inverse (denoted $\mathbf{P}, \mathbf{P}^{-1}$). The two programs $\mathcal{B}$ and $\mathcal{B}'$ are illustrated in Example 1.

Example 1. Two programs that compute the constant-zero function:

$$\begin{array}{rr|cccccc}
\mathcal{B} \;= & 0: & \mathbf{I} \ldots \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} \ldots \mathbf{I} \\
 & 1: & \mathbf{I} \ldots \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} \ldots \mathbf{I} \\
\hline
\mathcal{B}' = & 0: & \mathbf{I} \ldots \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} \ldots \mathbf{I} \\
 & 1: & \mathbf{I} \ldots \mathbf{I} & \mathbf{P} & \mathbf{I} & \mathbf{P}^{-1} & \mathbf{I} \ldots \mathbf{I} \\
\hline
\multicolumn{2}{r}{\text{Steps}:} & \mathcal{X} & u & v & w & \mathcal{Z} \\
\multicolumn{2}{r}{\text{input bits}:} & * \ldots * & j_1 & j_2 & j_1 & * \ldots *
\end{array} \tag{15}$$

The Attack. Recall that the GGHRSW obfuscator embeds the branching-program matrices $\mathbf{B}_{i,b}$ (and the identity for the dummy branch) into higher-dimension randomized matrices

$$\tilde{\mathbf{B}}_{i,b} := \begin{bmatrix} \mathbf{V}_{i,b} & \\ & \mathbf{B}_{i,b} \end{bmatrix} \text{ and } \tilde{\mathbf{B}'}_{i,b} := \begin{bmatrix} \mathbf{V'}_{i,b} & \\ & \mathbf{I} \end{bmatrix},$$

where $\mathbf{V}_{i,b}, \mathbf{V'}_{i,b}$ are random diagonal matrices. The $\tilde{\mathbf{B}}$'s are multiplied by the bundling scalars and Kilian randomization matrices, and then encoded to get

$$\mathbf{C}_{i,b} \;=\; \alpha_i^b \mathbf{K}_{i-1}^{-1} \tilde{\mathbf{B}}_{i,b} \mathbf{K}_i + g \cdot \mathbf{E}_{i,b} \;=\; \alpha_i^b \mathbf{K}_{i-1}^{-1} (\tilde{\mathbf{B}}_{i,b} + g \cdot \mathbf{F}_{i,b}) \mathbf{K}_i \pmod q \tag{16}$$

where $\mathbf{K}_{i-1}^{-1}$ is the inverse of $\mathbf{K}_{i-1}$ *modulo* $\langle g \rangle$, and $\mathbf{F}_{i,b}$ is the matrix satisfying $\alpha_i^b \mathbf{K}_{i-1}^{-1} \mathbf{F} \mathbf{K}_i = \mathbf{E} \pmod q$. (We ignore the denominator z in these notations, since it gets canceled when we apply zero-test.)

From Step II above we can obtain (some representatives of) the ratios β_1/β_0 and β_1'/β_0'. Namely, we can compute four scalars $\nu_0, \nu_1, \gamma_{00}, \gamma_{11} \in R$ such that

$$\frac{\nu_1}{\nu_0} = \frac{\beta_1'}{\beta_0'} \pmod{\mathcal{I}}, \quad \text{and} \quad \frac{\gamma_{11}}{\gamma_{00}} = \frac{\beta_1 \beta_1'}{\beta_0 \beta_0'} \pmod{\mathcal{I}}. \tag{17}$$

(Note that we chose notations that resemble their meaning: The scalars ν_0, ν_1 relate to the step v in the program, and γ_{00}, γ_{11} relate to the product of all relevant steps in the y interval.) Consider some values $x^{(i)} \in \{0,1\}^{|J_x|}$ for the bits that control steps in the $\mathcal{X}$ interval, τ, σ for the two distinguished bits that control steps in the $\mathcal{Y}$ interval, and $z^{(j)} \in \{0,1\}^{|J_z|}$ for the bits that control steps in the $\mathcal{Z}$ interval (all other bits are fixed). The resulting input is $u_{\sigma\tau}^{(i,j)} := x^{(i)}\sigma\tau z^{(j)}$, and it is a zero of the function. Also let $\mathsf{Eval}(u_{\sigma\tau}^{(i,j)})$ be the scalar obtained by evaluating the obfuscated branching program on this input:

$$\begin{aligned}\mathsf{Eval}(u) := \; & \frac{\eta}{g}\left(\mathbf{J}(\prod_{k\in[h]}\mathbf{C}_{k,u_{\iota(k)}})\mathbf{L} - \mathbf{J}'(\prod_{k\in[h]}\mathbf{C}'_{k,u_{\iota(k)}})\mathbf{L}'\right) \\ = \; & \frac{\eta}{g}\left(\prod_{k\in[h]}\alpha_{k,u_{\iota(k)}}\mathbf{J}\mathbf{L} - \prod_{k\in[h]}{\alpha'}_{k,u_{\iota(k)}}\mathbf{J}'\mathbf{L}' + g\cdot r_1(u) + g^2\cdot r_2(u) + \dots\right)\end{aligned} \tag{18}$$

where if u is a zero of the function then by construction we have

$$\prod_{k\in[h]}\alpha_{k,u_{\iota(k)}}\mathbf{J}\mathbf{L} - \prod_{k\in[h]}{\alpha'}_{k,u_{\iota(k)}}\mathbf{J}'\mathbf{L}' = 0.$$

In our attack, we choose many different $x^{(i)}$'s and $z^{(j)}$'s and for each i, j we compute

$$\begin{aligned}a_{i,j} := \quad & \mathsf{Eval}(x^{(i)}11z^{(j)})\cdot\gamma_{00}\cdot\nu_1\nu_0 \; - \; \mathsf{Eval}(x^{(i)}10z^{(j)})\cdot\gamma_{00}\cdot\nu_1\nu_1 \\ & - \mathsf{Eval}(x^{(i)}01z^{(j)})\cdot\gamma_{11}\cdot\nu_0\nu_0 \; + \; \mathsf{Eval}(x^{(i)}00z^{(j)})\cdot\gamma_{11}\cdot\nu_0\nu_1,\end{aligned} \tag{19}$$

where all the operations are carried out in the base ring R. Using sufficiently many $x^{(i)}$'s and $z^{(j)}$'s we get a matrix $\mathbf{A} = [a_{i,j}]_{i,j}$, and we check if this matrix has full rank modulo $\mathcal{I}$. We guess that the branching program is $\mathcal{B}'$ if $\mathbf{A}$ has full rank, and otherwise we guess that it is $\mathcal{B}$.

3.4 Analysis

The Matrix H We begin by considering the interval $\mathcal{Y}$ of the functional branch only. If $\mathcal{Y}$ consisted of only the steps u, v, w, then for any two bits $\sigma, \tau \in \{0,1\}$, the matrix that we get in the functional branch when evaluating on input with $u_{j_1} = \sigma$ and $u_{j_2} = \tau$ (namely $\mathbf{C}_{\sigma\tau} := \prod_{i\in\mathcal{Y}}\mathbf{C}_{i,u_{\iota(i)}}$) has the form

$$\begin{aligned}\mathbf{C}_{\sigma\tau} = \; & \beta_\sigma\beta'_\tau\cdot\mathbf{K}_{u-1}^{-1}\times\Bigg(\overbrace{\tilde{\mathbf{B}}_{u,\sigma}\tilde{\mathbf{B}}_{v,\tau}\tilde{\mathbf{B}}_{w,\sigma}}^{:=\tilde{\mathbf{B}}_{\mathcal{Y}}^{\sigma\tau}} + g\cdot\big(\tilde{\mathbf{B}}_{u,\sigma}\tilde{\mathbf{B}}_{v,\tau}(\overbrace{\mathbf{F}_{w,\sigma} + \mathbf{E}'_v\tilde{\mathbf{B}}_{w,\sigma}}^{:=\tilde{\mathbf{F}}_{w,\sigma}}) \\ & + \tilde{\mathbf{B}}_{u,\sigma}(\overbrace{\mathbf{F}_{v,\tau} + \mathbf{E}'_u\tilde{\mathbf{B}}_{v,\tau}}^{:=\tilde{\mathbf{F}}_{v,\tau}})\tilde{\mathbf{B}}_{w,\sigma} + \overbrace{\mathbf{F}_{u,\sigma}}^{:=\tilde{\mathbf{F}}_{u,\sigma}}\;\tilde{\mathbf{B}}_{v,\tau}\tilde{\mathbf{B}}_{w,\sigma}\big) + g^2\cdot\mathbf{E}_{\tau,\sigma}\Bigg)\times\mathbf{K}_w \\ = \; & \beta_\sigma\beta'_\tau\cdot\mathbf{K}_{u-1}^{-1}\times\big(\tilde{\mathbf{B}}_{\mathcal{Y}}^{\sigma\tau} + g\cdot\tilde{\mathbf{F}}_{\mathcal{Y}}^{\sigma\tau} + g^2\cdot\mathbf{E}_{\mathcal{Y}}^{\tau\sigma}\big)\times\mathbf{K}_w\end{aligned} \tag{20}$$

with equality modulo q, where $\mathbf{K}, \mathbf{K}^{-1}$'es are the Kilian randomization matrices, and $\mathbf{E}_{\mathcal{Y}}^{\tau\sigma}$ is some error matrix. (In the last line we have $\tilde{\mathbf{F}}_{\mathcal{Y}}^{\sigma\tau}$ denoting the coefficient of g in the $\mathcal{Y}$ interval.) If there are more steps in the interval $\mathcal{Y}$ then we get the same form, except the matrices $\tilde{\mathbf{B}}, \tilde{\mathbf{F}}$ are not single-step matrices but rather a product of a few steps, and we have an extra scalar factor α' (independent of the bits σ, τ) that comes from the bundling factors in the fixed steps in $\mathcal{Y}$.

The coefficient of g in Eqn. (20) is

$$\tilde{\mathbf{F}}_{\mathcal{Y}}^{\sigma\tau} := \tilde{\mathbf{B}}_{u,\sigma}\tilde{\mathbf{B}}_{v,\tau}\tilde{\mathbf{F}}_{w,\sigma} + \tilde{\mathbf{B}}_{u,\sigma}\tilde{\mathbf{F}}_{v,\tau}\tilde{\mathbf{B}}_{w,\sigma} + \tilde{\mathbf{F}}_{u,\sigma}\tilde{\mathbf{B}}_{v,\tau}\tilde{\mathbf{B}}_{w,\sigma}.$$

Let

$$\begin{aligned}
\mathbf{H} \;:=\; & \tilde{\mathbf{F}}_{\mathcal{Y}}^{11} - \tilde{\mathbf{F}}_{\mathcal{Y}}^{10} - \tilde{\mathbf{F}}_{\mathcal{Y}}^{01} + \tilde{\mathbf{F}}_{\mathcal{Y}}^{00} \\
= \; & \big(\tilde{\mathbf{B}}_{u,1}\tilde{\mathbf{B}}_{v,1}\tilde{\mathbf{F}}_{w,1} + \tilde{\mathbf{B}}_{u,1}\tilde{\mathbf{F}}_{v,1}\tilde{\mathbf{B}}_{w,1} + \tilde{\mathbf{F}}_{u,1}\tilde{\mathbf{B}}_{v,1}\tilde{\mathbf{B}}_{w,1}\big) \\
& -\big(\tilde{\mathbf{B}}_{u,1}\tilde{\mathbf{B}}_{v,0}\tilde{\mathbf{F}}_{w,1} + \tilde{\mathbf{B}}_{u,1}\tilde{\mathbf{F}}_{v,0}\tilde{\mathbf{B}}_{w,1} + \tilde{\mathbf{F}}_{u,1}\tilde{\mathbf{B}}_{v,0}\tilde{\mathbf{B}}_{w,1}\big) \\
& -\big(\tilde{\mathbf{B}}_{u,0}\tilde{\mathbf{B}}_{v,1}\tilde{\mathbf{F}}_{w,0} + \tilde{\mathbf{B}}_{u,0}\tilde{\mathbf{F}}_{v,1}\tilde{\mathbf{B}}_{w,0} + \tilde{\mathbf{F}}_{u,0}\tilde{\mathbf{B}}_{v,1}\tilde{\mathbf{B}}_{w,0}\big) \\
& +\big(\tilde{\mathbf{B}}_{u,0}\tilde{\mathbf{B}}_{v,0}\tilde{\mathbf{F}}_{w,0} + \tilde{\mathbf{B}}_{u,0}\tilde{\mathbf{F}}_{v,0}\tilde{\mathbf{B}}_{w,0} + \tilde{\mathbf{F}}_{u,0}\tilde{\mathbf{B}}_{v,0}\tilde{\mathbf{B}}_{w,0}\big).
\end{aligned} \tag{21}$$

The crux of the analysis is to show that $\mathbf{H}$ has a block of zeros when evaluating the program $\mathcal{B}$ (that has the identity matrices everywhere), but whp not when evaluating the branching program $\mathcal{B}'$ (that has $\mathbf{P}$ and $\mathbf{P}^{-1}$).

When evaluating $\mathcal{B}$, all the $\mathbf{B}_{i,b}$ matrices are the $w \times w$ identity $\mathbf{I}$, which are then embedded in the lower-right quadrant of the higher-dimension $\tilde{\mathbf{B}}_{i,b}$'s with the diagonal random $\mathbf{V}_i^b$'s in the upper-left quadrant. Below we also use the notation $\mathbf{V}_{ii'}^{\sigma\tau} := \mathbf{V}_i^{\sigma} \times \mathbf{V}_{i'}^{\tau}$ for the product of two of these diagonal matrices. We analyze separately the terms $\tilde{\mathbf{B}}\tilde{\mathbf{B}}\tilde{\mathbf{F}}$, $\tilde{\mathbf{B}}\tilde{\mathbf{F}}\tilde{\mathbf{B}}$, and $\tilde{\mathbf{F}}\tilde{\mathbf{B}}\tilde{\mathbf{B}}$, in order to establish that in this case the lower-right quadrant of $\mathbf{H}$ (that correspond to these identity matrices) is $\mathbf{0}$, i.e. $\mathbf{H} \in \begin{bmatrix} * & * \\ * & 0^{w\times w} \end{bmatrix}$.

(a) $\tilde{\mathbf{F}}\tilde{\mathbf{B}}\tilde{\mathbf{B}}$:

$$\begin{aligned}
& \tilde{\mathbf{F}}_u^1\tilde{\mathbf{B}}_v^1\tilde{\mathbf{B}}_w^1 - \tilde{\mathbf{F}}_u^1\tilde{\mathbf{B}}_v^0\tilde{\mathbf{B}}_w^1 - \tilde{\mathbf{F}}_u^0\tilde{\mathbf{B}}_v^1\tilde{\mathbf{B}}_w^0 + \tilde{\mathbf{F}}_u^0\tilde{\mathbf{B}}_v^0\tilde{\mathbf{B}}_w^0 \\
= \; & \tilde{\mathbf{F}}_u^1 \times \left(\begin{bmatrix} \mathbf{V}_{vw}^{11} & 0 \\ 0 & \mathbf{I} \end{bmatrix} - \begin{bmatrix} \mathbf{V}_{vw}^{01} & 0 \\ 0 & \mathbf{I} \end{bmatrix} \right) - \tilde{\mathbf{F}}_u^0 \times \left(\begin{bmatrix} \mathbf{V}_{vw}^{10} & 0 \\ 0 & \mathbf{I} \end{bmatrix} - \begin{bmatrix} \mathbf{V}_{vw}^{00} & 0 \\ 0 & \mathbf{I} \end{bmatrix} \right) \\
= \; & \tilde{\mathbf{F}}_u^1 \times \begin{bmatrix} \mathbf{V}_{vw}^{11} - \mathbf{V}_{vw}^{01} & 0 \\ 0 & 0 \end{bmatrix} - \tilde{\mathbf{F}}_u^0 \times \begin{bmatrix} \mathbf{V}_{vw}^{10} - \mathbf{V}_{vw}^{00} & 0 \\ 0 & 0 \end{bmatrix} \\
\in \; & \begin{bmatrix} *^{(2m+w)\times 2m}, & 0^{(2m+w)\times w} \end{bmatrix}
\end{aligned} \tag{22}$$

(b) $\tilde{\mathbf{B}}\tilde{\mathbf{B}}\tilde{\mathbf{F}}$:

$$\begin{aligned}
& \tilde{\mathbf{B}}_u^1\tilde{\mathbf{B}}_v^1\tilde{\mathbf{F}}_w^1 - \tilde{\mathbf{B}}_u^1\tilde{\mathbf{B}}_v^0\tilde{\mathbf{F}}_w^1 - \tilde{\mathbf{B}}_u^0\tilde{\mathbf{B}}_v^1\tilde{\mathbf{F}}_w^0 + \tilde{\mathbf{B}}_u^0\tilde{\mathbf{B}}_v^0\tilde{\mathbf{F}}_w^0 \\
= \; & \begin{bmatrix} \mathbf{V}_{uv}^{11} - \mathbf{V}_{uv}^{10} & 0 \\ 0 & 0 \end{bmatrix} \times \tilde{\mathbf{F}}_w^1 - \begin{bmatrix} \mathbf{V}_{uv}^{01} - \mathbf{V}_{uv}^{00} & 0 \\ 0 & 0 \end{bmatrix} \times \tilde{\mathbf{F}}_w^0 \in \begin{bmatrix} *^{2m\times(2m+w)} \\ 0^{w\times(2m+w)} \end{bmatrix}
\end{aligned} \tag{23}$$

(c) The most interesting term is $\tilde{\mathbf{B}}\tilde{\mathbf{F}}\tilde{\mathbf{B}}$:

$$\begin{aligned}
&\tilde{\mathbf{B}}^1_u\tilde{\mathbf{F}}^1_v\tilde{\mathbf{B}}^1_w - \tilde{\mathbf{B}}^1_u\tilde{\mathbf{F}}^0_v\tilde{\mathbf{B}}^1_w - \tilde{\mathbf{B}}^0_u\tilde{\mathbf{F}}^1_v\tilde{\mathbf{B}}^0_w + \tilde{\mathbf{B}}^0_u\tilde{\mathbf{F}}^0_v\tilde{\mathbf{B}}^0_w \\
&= \begin{bmatrix} * & * \\ * \; \mathbf{I} & \widetilde{\mathbf{F}^{\mathbf{1}}}_{\mathbf{v}(LR)} \; \mathbf{I} \end{bmatrix} - \begin{bmatrix} * & * \\ * \; \mathbf{I} & \widetilde{\mathbf{F}^{\mathbf{0}}}_{\mathbf{v}(LR)} \; \mathbf{I} \end{bmatrix} - \begin{bmatrix} * & * \\ * \; \mathbf{I} & \widetilde{\mathbf{F}^{\mathbf{1}}}_{\mathbf{v}(LR)} \; \mathbf{I} \end{bmatrix} + \begin{bmatrix} * & * \\ * \; \mathbf{I} & \widetilde{\mathbf{F}^{\mathbf{0}}}_{\mathbf{v}(LR)} \; \mathbf{I} \end{bmatrix} \quad (24) \\
&\in \begin{bmatrix} *^{2m\times 2m}, & *^{2m\times w} \\ *^{w\times 2m}, & 0^{w\times w} \end{bmatrix}
\end{aligned}$$

where the subscript $\tilde{\mathbf{F}}_{(LR)}$ denotes the lower-right quadrant (of dimension $w \times w$) in the corresponding matrix.

Adding Equations (22), (24) and (23), we get $\mathbf{H} \in \begin{bmatrix} * & * \\ * & 0^{w\times w} \end{bmatrix}$, as needed.

When evaluating $\mathcal{B}'$, the form of the terms $\tilde{\mathbf{B}}\tilde{\mathbf{F}}\tilde{\mathbf{B}}$ changes: Instead of Eqn. (24), in the lower-right quadrant we now get $\mathbf{H}_{(LR)} = \widetilde{\mathbf{F}^{\mathbf{1}}}_{\mathbf{v}(LR)} - \widetilde{\mathbf{F}^{\mathbf{0}}}_{\mathbf{v}(LR)} - \mathbf{P}(\widetilde{\mathbf{F}^{\mathbf{1}}}_{\mathbf{v}(LR)} - \widetilde{\mathbf{F}^{\mathbf{0}}}_{\mathbf{v}(LR)})\mathbf{P}^{-1}$, which is unlikely to be the zero matrix.

The same analysis can be applied to the dummy branch, where we can define the matrix $\mathbf{H}'$ in the same way. In the dummy branch, however, the lower-right quadrant of $\mathbf{H}'$ is always zero, in both $\mathcal{B}$ and $\mathcal{B}'$ (since the dummy branch always consists of identity matrices, regardless of what the program is).

The Matrix A We now proceed to incorporate the $\mathcal{X}, \mathcal{Z}$ intervals (including the bookends) and analyze the matrix $\mathbf{A} = [a_{i,j}]_{i,j}$. For any fixed i, j, let us denote the product of the $\mathcal{X}$ interval matrices in the two branches (including the bookend) by $\alpha_x^{(i)} \cdot \mathbf{J}(\tilde{\mathbf{B}}^{(i)}_{\mathcal{X}} + g \cdot \tilde{\mathbf{F}}^{(i)}_{\mathcal{X}})$ and $\alpha'^{(i)}_x \cdot \mathbf{J}'(\tilde{\mathbf{B}}'^{(i)}_{\mathcal{X}} + g \cdot \tilde{\mathbf{F}}'^{(i)}_{\mathcal{X}})$, respectively. Similarly for the $\mathcal{Z}$ interval we denote the products in the two branches by $\alpha_z^{(j)}(\tilde{\mathbf{B}}^{(j)}_{\mathcal{Z}} + g \cdot \tilde{\mathbf{F}}^{(j)}_{\mathcal{Z}})\mathbf{L}$ and $\alpha'^{(j)}_z(\tilde{\mathbf{B}}'^{(j)}_{\mathcal{Z}} + g \cdot \tilde{\mathbf{F}}'^{(j)}_{\mathcal{Z}})\mathbf{L}'$, respectively.

By construction — for the case where the $\mathcal{Y}$ interval includes just the steps u, v, w — we have $\alpha_x^{(i)}\alpha_z^{(j)} = \alpha'^{(i)}_x\alpha'^{(j)}_z$, and we denote this product by $\alpha^{(i,j)}$. (In the more general case we have the same equality, except it includes also the constants α_y, α'_y due to the fixed steps in the $\mathcal{Y}$ interval.) With these notations, we have

$$\begin{aligned}
&\mathsf{Eval}\left(x^{(i)}\sigma\tau z^{(j)}\right) \\
&= \alpha^{(i,j)}\beta_\sigma\beta'_\tau \cdot \frac{\eta}{g}\Big(\mathbf{J}\big(\tilde{\mathbf{B}}^{(i)}_{\mathcal{X}} + g \cdot \tilde{\mathbf{F}}^{(i)}_{\mathcal{X}}\big)\big(\tilde{\mathbf{B}}^{\sigma\tau}_{\mathcal{Y}} + g \cdot \tilde{\mathbf{F}}^{\sigma\tau}_{\mathcal{Y}}\big)\big(\tilde{\mathbf{B}}^{(j)}_{\mathcal{Z}} + g \cdot \tilde{\mathbf{F}}^{(j)}_{\mathcal{Z}}\big)\mathbf{L} \\
&\quad - \mathbf{J}'\big(\tilde{\mathbf{B}}'^{(i)}_{\mathcal{X}} + g \cdot \tilde{\mathbf{F}}'^{(i)}_{\mathcal{X}}\big)\big(\tilde{\mathbf{B}}'^{\sigma\tau}_{\mathcal{Y}} + g \cdot \tilde{\mathbf{F}}'^{\sigma\tau}_{\mathcal{Y}}\big)\big(\tilde{\mathbf{B}}'^{(j)}_{\mathcal{Z}} + g \cdot \tilde{\mathbf{F}}'^{(j)}_{\mathcal{Z}}\big)\mathbf{L}'\Big) \qquad (25) \\
&= \alpha^{(i,j)}\beta_\sigma\beta'_\tau \cdot \eta\Big(\mathbf{J}\left(\tilde{\mathbf{B}}^{(i)}_{\mathcal{X}}\tilde{\mathbf{B}}^{\sigma\tau}_{\mathcal{Y}}\tilde{\mathbf{F}}^{(j)}_{\mathcal{Z}} + \tilde{\mathbf{B}}^{(i)}_{\mathcal{X}}\tilde{\mathbf{F}}^{\sigma\tau}_{\mathcal{Y}}\tilde{\mathbf{B}}^{(j)}_{\mathcal{Z}} + \tilde{\mathbf{F}}^{(i)}_{\mathcal{X}}\tilde{\mathbf{B}}^{\sigma\tau}_{\mathcal{Y}}\tilde{\mathbf{B}}^{(j)}_{\mathcal{Z}}\right)\mathbf{L} \\
&\quad - \mathbf{J}'\left(\tilde{\mathbf{B}}'^{(i)}_{\mathcal{X}}\tilde{\mathbf{B}}'^{\sigma\tau}_{\mathcal{Y}}\tilde{\mathbf{F}}'^{(j)}_{\mathcal{Z}} + \tilde{\mathbf{B}}'^{(i)}_{\mathcal{X}}\tilde{\mathbf{F}}'^{\sigma\tau}_{\mathcal{Y}}\tilde{\mathbf{B}}'^{(j)}_{\mathcal{Z}} + \tilde{\mathbf{F}}'^{(i)}_{\mathcal{X}}\tilde{\mathbf{B}}'^{\sigma\tau}_{\mathcal{Y}}\tilde{\mathbf{B}}'^{(j)}_{\mathcal{Z}}\right)\mathbf{L}'\Big) \pmod{\mathcal{I}}
\end{aligned}$$

where the last equality follows since $x^{(i)}\sigma\tau z^{(j)}$ is a zero of the function, and hence the "free term" without any factor of g is equal to zero. Using Eqn. (26) we can re-write $a_{i,j}$ as

$$\begin{aligned} a_{i,j} &= \alpha^{(i,j)}\beta_1\beta_1'\eta(\cdots)\gamma_{00}\nu_1\nu_0 \;-\; \alpha^{(i,j)}\beta_1\beta_0'\eta(\cdots)\gamma_{00}\nu_1\nu_1 \\ &\quad - \alpha^{(i,j)}\beta_0\beta_1'\eta(\cdots)\gamma_{11}\nu_0\nu_0 \;+\; \alpha^{(i,j)}\beta_0\beta_0'\eta(\cdots)\gamma_{11}\nu_0\nu_1 \pmod{\mathcal{I}} \end{aligned}$$

where the $(\cdots)$'s refer to the parenthesized expression from Eqn. (26) relative to the appropriate bits σ, τ. This is where we use the ratios that we recovered in Step II, by definition we have that

$$\beta_1\beta_1'\cdot\gamma_{00}\nu_1\nu_0 = \beta_1\beta_0'\cdot\gamma_{00}\nu_1\nu_1 = \beta_0\beta_1'\cdot\gamma_{11}\nu_0\nu_0 = \beta_0\beta_0'\cdot\gamma_{11}\nu_0\nu_1 \pmod{\mathcal{I}},$$

so the four terms above (with i, j fixed) all have the same scalar multiple. Moreover that scalar is bilinear in i, j, so we just fold it into the matrices corresponding to $x^{(i)}, z^{(j)}$ and ignore it from now on. Thus we can further re-write the expression for $a_{i,j}$ as

$$\begin{aligned} a_{i,j} = \;& \mathbf{J}\Big(\tilde{\mathbf{B}}_{\mathcal{X}}^{(i)}\left(\tilde{\mathbf{B}}_{\mathcal{Y}}^{11} - \tilde{\mathbf{B}}_{\mathcal{Y}}^{10} - \tilde{\mathbf{B}}_{\mathcal{Y}}^{01} + \tilde{\mathbf{B}}_{\mathcal{Y}}^{00}\right)\tilde{\mathbf{F}}_{\mathcal{Z}}^{(j)} \\ & \quad + \tilde{\mathbf{B}}_{\mathcal{X}}^{(i)}\overbrace{\left(\tilde{\mathbf{F}}_{\mathcal{Y}}^{11} - \tilde{\mathbf{F}}_{\mathcal{Y}}^{10} - \tilde{\mathbf{F}}_{\mathcal{Y}}^{01} + \tilde{\mathbf{F}}_{\mathcal{Y}}^{00}\right)}^{=\mathbf{H}}\tilde{\mathbf{B}}_{\mathcal{Z}}^{(j)} \\ & \quad + \tilde{\mathbf{F}}_{\mathcal{X}}^{(i)}\left(\tilde{\mathbf{B}}_{\mathcal{Y}}^{11} - \tilde{\mathbf{B}}_{\mathcal{Y}}^{10} - \tilde{\mathbf{B}}_{\mathcal{Y}}^{01} + \tilde{\mathbf{B}}_{\mathcal{Y}}^{00}\right)\tilde{\mathbf{B}}_{\mathcal{Z}}^{(j)}\Big)\mathbf{L} \\ & - \mathbf{J}'\Big(\tilde{\mathbf{B}}'^{(i)}_{\mathcal{X}}\left(\tilde{\mathbf{B}}'^{11}_{\mathcal{Y}} - \tilde{\mathbf{B}}'^{10}_{\mathcal{Y}} - \tilde{\mathbf{B}}'^{01}_{\mathcal{Y}} + \tilde{\mathbf{B}}'^{00}_{\mathcal{Y}}\right)\tilde{\mathbf{F}}'^{(j)}_{\mathcal{Z}} \\ & \quad + \tilde{\mathbf{B}}'^{(i)}_{\mathcal{X}}\overbrace{\left(\tilde{\mathbf{F}}'^{11}_{\mathcal{Y}} - \tilde{\mathbf{F}}'^{10}_{\mathcal{Y}} - \tilde{\mathbf{F}}'^{01}_{\mathcal{Y}} + \tilde{\mathbf{F}}'^{00}_{\mathcal{Y}}\right)}^{=\mathbf{H}'}\tilde{\mathbf{B}}'^{(j)}_{\mathcal{Z}} \\ & \quad + \tilde{\mathbf{F}}'^{(i)}_{\mathcal{X}}\left(\tilde{\mathbf{B}}'^{11}_{\mathcal{Y}} - \tilde{\mathbf{B}}'^{10}_{\mathcal{Y}} - \tilde{\mathbf{B}}'^{01}_{\mathcal{Y}} + \tilde{\mathbf{B}}'^{00}_{\mathcal{Y}}\right)\tilde{\mathbf{B}}'^{(j)}_{\mathcal{Z}}\Big)\mathbf{L}' \pmod{\mathcal{I}} \end{aligned} \tag{26}$$

Next, we denote:

$$\begin{aligned} &\tilde{\mathbf{B}}_{\mathcal{Y}}^{\Delta} := \tilde{\mathbf{B}}_{\mathcal{Y}}^{11} - \tilde{\mathbf{B}}_{\mathcal{Y}}^{10} - \tilde{\mathbf{B}}_{\mathcal{Y}}^{01} + \tilde{\mathbf{B}}_{\mathcal{Y}}^{00},\; \tilde{\mathbf{B}}'^{\Delta}_{\mathcal{Y}} := \tilde{\mathbf{B}}'^{11}_{\mathcal{Y}} - \tilde{\mathbf{B}}'^{10}_{\mathcal{Y}} - \tilde{\mathbf{B}}'^{01}_{\mathcal{Y}} + \tilde{\mathbf{B}}'^{00}_{\mathcal{Y}} \\ &\mathbf{x}_i := \mathbf{J}\tilde{\mathbf{B}}_{\mathcal{X}}^{(i)}, \qquad \mathbf{z}_j := \tilde{\mathbf{B}}_{\mathcal{Z}}^{(j)}\mathbf{L}, \quad \mathbf{x}_i' := \mathbf{J}'\tilde{\mathbf{B}}'^{(i)}_{\mathcal{X}}, \qquad \mathbf{z}_j' := \tilde{\mathbf{B}}'^{(j)}_{\mathcal{Z}}\mathbf{L}', \\ &\mathbf{e}_i := \mathbf{J}\tilde{\mathbf{F}}_{\mathcal{X}}^{(i)}, \qquad \mathbf{f}_j := \tilde{\mathbf{F}}_{\mathcal{Z}}^{(j)}\mathbf{L}, \quad \mathbf{e}_i' := \mathbf{J}'\tilde{\mathbf{F}}'^{(i)}_{\mathcal{X}}, \qquad \mathbf{f}_j' := \tilde{\mathbf{F}}'^{(j)}_{\mathcal{Z}}\mathbf{L}' \end{aligned}$$

and so we can write

$$a_{i,j} = \underbrace{\mathbf{x}_i\tilde{\mathbf{B}}_{\mathcal{Y}}^{\Delta}\mathbf{f}_j + \mathbf{x}_i\mathbf{H}\mathbf{z}_j + \mathbf{e}_i\tilde{\mathbf{B}}_{\mathcal{Y}}^{\Delta}\mathbf{z}_j}_{:=d_{i,j}} - \underbrace{\mathbf{x}_i'\tilde{\mathbf{B}}'^{\Delta}_{\mathcal{Y}}\mathbf{f}_j' + \mathbf{x}_i'\mathbf{H}'\mathbf{z}_j' + \mathbf{e}_i'\tilde{\mathbf{B}}'^{\Delta}_{\mathcal{Y}}\mathbf{z}_j'}_{:=d'_{i,j}} \pmod{\mathcal{I}}. \tag{27}$$

Denoting $\mathbf{D} = [d_{i,j}]_{i,j}$ and $\mathbf{D}' = [d'_{i,j}]_{i,j}$, we have $\mathbf{A} = \mathbf{D} - \mathbf{D}'$, and so the rank of $\mathbf{A}$ is at most $\mathsf{rank}(\mathbf{D}) + \mathsf{rank}(\mathbf{D}')$. Recalling the structure of the various components again, we note that they contain many zeros. In particular for the program $\mathcal{B}$ we have $\mathbf{x}_i, \mathbf{x}'_i \in (0^m \ \ *^m \ \ *^w)$, $\mathbf{z}_j, \mathbf{z}'_j \in (*^m \ 0^m \ *^w)^t$, and also

$$\tilde{\mathbf{B}}^{\varDelta}_{\mathcal{Y}},\ \tilde{\mathbf{B}}'^{\varDelta}_{\mathcal{Y}} \in \begin{pmatrix} *^{m\times m}, 0^{m\times m}, 0^{m\times w}\\ 0^{m\times m}, *^{m\times m}, 0^{m\times w}\\ 0^{w\times m}, 0^{w\times m}, 0^{w\times w}\end{pmatrix}, \quad \mathbf{H},\mathbf{H}' \in \begin{pmatrix} *^{m\times m}, *^{m\times m}, *^{m\times w}\\ *^{m\times m}, *^{m\times m}, *^{m\times w}\\ *^{w\times m}, *^{w\times m}, 0^{w\times w}\end{pmatrix},$$

and for $\mathcal{B}'$ we have almost the same thing except that $\mathbf{H}$ can be arbitrary. Our goal is to detect this difference in the form of $\mathbf{H}$ given sufficiently many $a_{i,j}$'s. Let us analyze first only the term from the functional branch, $\mathbf{D} = [d_{i,j}]_{i,j}$, where we pick $\zeta \geq 2m+1$ different i's and j's.

$$\begin{aligned}\mathbf{D} = {} & \mathbf{X}\tilde{\mathbf{B}}^{\varDelta}_{\mathcal{Y}}\mathbf{F} + \mathbf{XHZ} + \mathbf{E}\tilde{\mathbf{B}}^{\varDelta}_{\mathcal{Y}}\mathbf{Z}\\ = {} & \begin{bmatrix}0\ \mathbf{X}_2\ \mathbf{X}_3\end{bmatrix}\begin{bmatrix}\tilde{\mathbf{B}}_{1,1} & 0 & 0\\ 0 & \tilde{\mathbf{B}}_{2,2} & 0\\ 0 & 0 & 0\end{bmatrix}\begin{bmatrix}\mathbf{F}_1\\ \mathbf{F}_2\\ \mathbf{F}_3\end{bmatrix} + \begin{bmatrix}0\ \mathbf{X}_2\ \mathbf{X}_3\end{bmatrix}\begin{bmatrix}\mathbf{H}_{1,1} & \mathbf{H}_{1,2} & \mathbf{H}_{1,3}\\ \mathbf{H}_{2,1} & \mathbf{H}_{2,2} & \mathbf{H}_{2,3}\\ \mathbf{H}_{3,1} & \mathbf{H}_{3,2} & 0\end{bmatrix}\begin{bmatrix}\mathbf{Z}_1\\ 0\\ \mathbf{Z}_3\end{bmatrix}\\ & + \begin{bmatrix}\mathbf{E}_1\ \mathbf{E}_2\ \mathbf{E}_3\end{bmatrix}\begin{bmatrix}\tilde{\mathbf{B}}_{1,1} & 0 & 0\\ 0 & \tilde{\mathbf{B}}_{2,2} & 0\\ 0 & 0 & 0\end{bmatrix}\begin{bmatrix}\mathbf{Z}_1\\ 0\\ \mathbf{Z}_3\end{bmatrix}\end{aligned} \tag{28}$$

where $\{\tilde{\mathbf{B}}_{k,\ell}, \mathbf{H}_{k,\ell}\}_{k,\ell\in[3]}$ are blocks of $\tilde{\mathbf{B}}^{\varDelta}_{\mathcal{Y}}, \mathbf{H}$ with dimensions $[m|m|w]\times[m|m|w]$. $\{\mathbf{X}_k, \mathbf{E}_k\}_{k\in[3]}$ are blocks of $\mathbf{X}$, $\mathbf{E}$ with dimensions $\zeta \times [m|m|w]$. $\{\mathbf{Z}_\ell, \mathbf{F}_\ell\}_{\ell\in[3]}$ are blocks of $\mathbf{Z}$, $\mathbf{F}$ with dimensions $[m|m|w] \times \zeta$.

Observe that many of the blocks in Eqn. (28) do not contribute to the result, since they are only multiplied by zeros in the adjacent matrices. For example, E_3 in the last term above does not contribute to the evaluation since the entries in the 3^{rd} blocked rows of $\tilde{\mathbf{B}}^{\varDelta}_{\mathcal{Y}}$ are all zeros. We can therefore treat these blocks as if they were zeros themselves, so we get

$$\begin{aligned}\mathbf{D} = {} & \begin{bmatrix}0\ \mathbf{X}_2\ 0\end{bmatrix}\begin{bmatrix}0 & 0 & 0\\ 0 & \tilde{\mathbf{B}}_{2,2} & 0\\ 0 & 0 & 0\end{bmatrix}\begin{bmatrix}0\\ \mathbf{F}_2\\ 0\end{bmatrix} + \begin{bmatrix}0\ \mathbf{X}_2\ \mathbf{X}_3\end{bmatrix}\begin{bmatrix}0 & 0 & 0\\ \mathbf{H}_{2,1} & 0 & \mathbf{H}_{2,3}\\ \mathbf{H}_{3,1} & 0 & 0\end{bmatrix}\begin{bmatrix}\mathbf{Z}_1\\ 0\\ \mathbf{Z}_3\end{bmatrix}\\ & + \begin{bmatrix}\mathbf{E}_1\ 0\ 0\end{bmatrix}\begin{bmatrix}\tilde{\mathbf{B}}_{1,1} & 0 & 0\\ 0 & 0 & 0\\ 0 & 0 & 0\end{bmatrix}\begin{bmatrix}\mathbf{Z}_1\\ 0\\ 0\end{bmatrix}\end{aligned} \tag{29}$$

From there we get

$$\begin{aligned}\mathbf{D} = {} & \mathbf{X}_2\tilde{\mathbf{B}}_{2,2}\mathbf{F}_2 + \begin{bmatrix}\mathbf{X}_2\mathbf{H}_{2,1} + \mathbf{X}_3\mathbf{H}_{3,1}\ 0\ \mathbf{X}_2\mathbf{H}_{2,3}\end{bmatrix}\begin{bmatrix}\mathbf{Z}_1\\ 0\\ \mathbf{Z}_3\end{bmatrix} + \mathbf{E}_1\tilde{\mathbf{B}}_{1,1}\mathbf{Z}_1\\ = {} & \mathbf{X}_2\tilde{\mathbf{B}}_{2,2}\mathbf{F}_2 + (\mathbf{X}_2\mathbf{H}_{2,1} + \mathbf{X}_3\mathbf{H}_{3,1})\mathbf{Z}_1 + \mathbf{X}_2\mathbf{H}_{2,3}\mathbf{Z}_3 + \mathbf{E}_1\tilde{\mathbf{B}}_{1,1}\mathbf{Z}_1\\ = {} & \mathbf{X}_2(\tilde{\mathbf{B}}_{2,2}\mathbf{F}_2 + \mathbf{H}_{2,3}\mathbf{Z}_3) + (\mathbf{X}_2\mathbf{H}_{2,1} + \mathbf{X}_3\mathbf{H}_{3,1} + \mathbf{E}_1\tilde{\mathbf{B}}_{1,1})\mathbf{Z}_1\end{aligned} \tag{30}$$

The ranks of block matrices $\mathbf{X}_2$ and $\mathbf{Z}_1$ are upper-bounded by m, which means $\mathbf{D}$ is the sum of two matrices of rank m, hence the maximum rank is $2m$.

For $\mathcal{B}'$, the rank of $\mathbf{D}$ is $2m+1$ whp. To see the difference in the analysis, in Eqn. (29) the potential $\mathbf{H}_{3,3}$ block is non-zero, so whp $\mathbf{D}$ is not decomposable to the sum of 2 matrices of rank m like for $\mathcal{B}$.

The analysis of $\mathbf{D}'$ for both $\mathcal{B}$ and $\mathcal{B}'$ is analogous to the analysis of $\mathbf{D}$ in $\mathcal{B}$, i.e. in both cases the rank of $\mathbf{D}'$ is at most $2m$. So we are able to distinguish $\mathcal{B}$ and $\mathcal{B}'$ by obtaining $\mathbf{A} = [a_{i,j}]_{i,j}$ from picking $\zeta \geq 4m+1$ different i's and j's, and computing the rank of $\mathbf{A}$.

3.5 Discussions of Recent Immunizations

Recently Garg et al. [24] (merged from two similar proposals [25,36]) propose immunization mechanisms against the annihilation attack. The common feature of the immunizations is to pad random $2m$-by-$2m$ matrices instead of entries on the diagonal (i.e. change the matrices $\mathbf{V}_{i,b}$ and $\mathbf{V'}_{i,b}$ in Eqn. (1) from diagonal to fully random), so as to encode a pseudorandom function in the noises. The difference of the two proposals lies in the ways to instantiate the paradigm.

We observe that the immunizations do not stop the attack if the branching program is input-partitioning. The observation does not contradict the proofs of security in the weakened idealized model from [24], since they require dual-input branching programs (which are not input-partitioning).

Below we briefly describe the two immunizations. In the immunization proposed by Miles, Sahai and Zhandry [36], the bookend vectors are changed to

$$\mathbf{J}, \mathbf{J}' \in \left[0^{2m}, \$^{w}\right], \quad \mathbf{L}, \mathbf{L}' \in \left[\$^{2m}, \$^{w}\right]^{T} \tag{31}$$

These changes do not affect the algorithms and analyses in Steps I and II. In Step III, the analysis of the matrix $\mathbf{H}$ in Eqn. (21) remains the same. The analysis of the rank of $\mathbf{D}$ in Eqn. (28) changes slightly. For program $\mathcal{B}$ in Example 1,

$$\begin{aligned} \mathbf{D} &= \mathbf{X}\tilde{\mathbf{B}}^{\Delta}_{\mathcal{Y}}\mathbf{F} + \mathbf{X}\mathbf{H}\mathbf{Z} + \mathbf{E}\tilde{\mathbf{B}}^{\Delta}_{\mathcal{Y}}\mathbf{Z} \\ &= \begin{bmatrix} 0 & \mathbf{X}_2 \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{B}}_{1,1} & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{F}_1 \\ \mathbf{F}_2 \end{bmatrix} + \begin{bmatrix} 0 & \mathbf{X}_2 \end{bmatrix} \begin{bmatrix} \mathbf{H}_{1,1} & \mathbf{H}_{1,2} \\ \mathbf{H}_{2,1} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{Z}_1 \\ \mathbf{Z}_2 \end{bmatrix} + \begin{bmatrix} \mathbf{E}_1 & \mathbf{E}_2 \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{B}}_{1,1} & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{Z}_1 \\ \mathbf{Z}_2 \end{bmatrix} \\ &= 0 + \mathbf{X}_2\mathbf{H}_{2,1}\mathbf{Z}_1 + \mathbf{E}_1\tilde{\mathbf{B}}_{1,1}\mathbf{Z}_1 = (\mathbf{X}_2\mathbf{H}_{2,1} + \mathbf{E}_1\tilde{\mathbf{B}}_{1,1})\mathbf{Z}_1 \end{aligned} \tag{32}$$

where $\{\tilde{\mathbf{B}}_{k,\ell}, \mathbf{H}_{k,\ell}\}_{k,\ell\in[2]}$ are blocks of $\tilde{\mathbf{B}}^{\Delta}_{\mathcal{Y}}$, $\mathbf{H}$ with dimensions $[2m|w] \times [2m|w]$. $\{\mathbf{X}_k, \mathbf{E}_k\}_{k\in[2]}$ are blocks of $\mathbf{X}$, $\mathbf{E}$ with dimensions $\zeta \times [2m|w]$. $\{\mathbf{Z}_\ell, \mathbf{F}_\ell\}_{\ell\in[2]}$ are blocks of $\mathbf{Z}$, $\mathbf{F}$ with dimensions $[2m|w] \times \zeta$. The rank of $\mathbf{D}$ is thus upper-bounded by $2m$. For program $\mathcal{B}'$, the potential non-zero $\mathbf{H}_{2,2}$ contributes to an additional term $\mathbf{X}_2\mathbf{H}_{2,2}\mathbf{Z}_2$. So the same algorithm from Section 3.3 distinguishes $\mathcal{B}$ and $\mathcal{B}'$.

More changes are made in the immunization proposed by Garg, Mukherjee and Srinivasan [25]. The plaintext space is set to be $R/\mathcal{J}$ where $\mathcal{J} = \langle g^2 \rangle$. The encoding of $s \in R/\mathcal{J}$ is a short representative of the coset $s + \mathbf{J}$. The zero-test parameter remains the same: $p_{zt} = \eta z^{\kappa}/g$. The bookend vectors are changed to

$$\begin{aligned}\mathbf{J}, \mathbf{J}' &:= \left[g \cdot \mathbf{J}_1, \mathbf{J}_2\right], \left[g \cdot \mathbf{J}'_1, \mathbf{J}'_2\right] \in \left[g \cdot \$^{2m}, \$^{w}\right] \\ \mathbf{L}, \mathbf{L}' &:= \left[\mathbf{L}_1, \mathbf{L}_2\right]^T, \left[\mathbf{L}'_1, \mathbf{L}'_2\right]^T \in \left[\$^{2m}, \$^{w}\right]^T.\end{aligned} \tag{33}$$

where $\mathbf{J}_2\mathbf{L}_2 = \mathbf{J}'_2\mathbf{L}'_2$. An honest evaluation analogous to Eqn. (18) can be expressed as

$$\begin{aligned}\mathsf{Eval}(u) :=&\ \frac{\eta}{g}\left(\mathbf{J}(\prod_{k\in[h]} \mathbf{C}_{k,u_{\iota(k)}})\mathbf{L} - \mathbf{J}'(\prod_{k\in[h]} \mathbf{C}'_{k,u_{\iota(k)}})\mathbf{L}'\right) \\ =&\ \frac{\eta}{g}\left(\prod_{k\in[h]} \alpha_{k,u_{\iota(k)}}\mathbf{J}_2\mathbf{L}_2 - \prod_{k\in[h]} \alpha'_{k,u_{\iota(k)}}\mathbf{J}'_2\mathbf{L}'_2 + g \cdot r_1(u) + g^2 \cdot r_2(u) + \dots\right)\end{aligned} \tag{34}$$

where if u is a zero of the function then

$$\prod_{k\in[h]} \alpha_{k,u_{\iota(k)}}\mathbf{J}_2\mathbf{L}_2 - \prod_{k\in[h]} \alpha'_{k,u_{\iota(k)}}\mathbf{J}'_2\mathbf{L}'_2 = 0.$$

The immunization changes the coefficient of g^1 into

$$r_1 = \left(\mathbf{J}_1 \prod_{k\in[h]} \mathbf{V}_{k,u_{\iota(k)}}\mathbf{L}_1 - \mathbf{J}'_1 \prod_{k\in[h]} \mathbf{V}'_{k,u_{\iota(k)}}\mathbf{L}'_1\right),$$

and pushes all the information about the secrets up to the coefficients of higher order terms. This is the rationale of Garg, Mukherjee and Srinivasan's [25] immunization against annihilation attacks.

Still, for branching programs with input-partitioning, these immunizations do not affect the algorithms and the analyses in Steps I and II, except that we obtain a basis of $\langle g^2 \rangle$ and (possibly big) representatives of scalars α in the coset $\alpha + \langle g^2 \rangle$. In Step III, we analyze $\mathbf{H}$ and $\mathbf{A}$ modulo $\mathcal{J}$ instead of modulo $\mathcal{I}$. The feature of $\mathbf{H}$ remains the same. To $\mathbf{A}$, the expression of each $a_{i,j}$ from Eqn. (27) shall be modified to (the following expression still contains coefficients of g^2 that will be removed later)

$$\begin{aligned}a_{i,j} =&\ \underbrace{\frac{\eta}{g}\left(\mathbf{x}_i\tilde{\mathbf{B}}^{\varDelta}_{\mathcal{Y}}\mathbf{z}_j + g^2 \cdot \left(\mathbf{x}_i\tilde{\mathbf{B}}^{\varDelta}_{\mathcal{Y}}\mathbf{f}_j + \mathbf{x}_i\mathbf{H}\mathbf{z}_j + \mathbf{e}_i\tilde{\mathbf{B}}^{\varDelta}_{\mathcal{Y}}\mathbf{z}_j\right)\right)}_{:=d_{i,j}} \\ &- \underbrace{\frac{\eta}{g}\left(\mathbf{x}'_i\tilde{\mathbf{B}}'^{\varDelta}_{\mathcal{Y}}\mathbf{z}'_j + g^2\left(\mathbf{x}'_i\tilde{\mathbf{B}}'^{\varDelta}_{\mathcal{Y}}\mathbf{f}'_j + \mathbf{x}'_i\mathbf{H}'\mathbf{z}'_j + \mathbf{e}'_i\tilde{\mathbf{B}}'^{\varDelta}_{\mathcal{Y}}\mathbf{z}'_j\right)\right)}_{:=d'_{i,j}} \pmod{\mathcal{J}}.\end{aligned} \tag{35}$$

Examining the functional component $\mathbf{D}$ for $\mathcal{B}$, with the same blocked dimensions as Eqn. (32):

$$\begin{aligned}
\mathbf{D} = {} & \frac{\eta}{g}\left(\mathbf{X}\tilde{\mathbf{B}}^{\varDelta}_{\mathcal{Y}}\mathbf{Z} + g^2 \cdot \left(\mathbf{X}\tilde{\mathbf{B}}^{\varDelta}_{\mathcal{Y}}\mathbf{F} + \mathbf{X}\mathbf{H}\mathbf{Z} + \mathbf{E}\tilde{\mathbf{B}}^{\varDelta}_{\mathcal{Y}}\mathbf{Z}\right)\right) \\
= {} & \frac{\eta}{g}\begin{bmatrix} g\mathbf{X}_1 & \mathbf{X}_2 \end{bmatrix}\begin{bmatrix} \tilde{\mathbf{B}}_{1,1} & 0 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} \mathbf{Z}_1 \\ \mathbf{Z}_2 \end{bmatrix} + \eta g\Bigg(\begin{bmatrix} g\mathbf{X}_1 & \mathbf{X}_2 \end{bmatrix}\begin{bmatrix} \tilde{\mathbf{B}}_{1,1} & 0 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} \mathbf{F}_1 \\ \mathbf{F}_2 \end{bmatrix} \\
& \qquad + \begin{bmatrix} g\mathbf{X}_1 & \mathbf{X}_2 \end{bmatrix}\begin{bmatrix} \mathbf{H}_{1,1} & \mathbf{H}_{1,2} \\ \mathbf{H}_{2,1} & 0 \end{bmatrix}\begin{bmatrix} \mathbf{Z}_1 \\ \mathbf{Z}_2 \end{bmatrix} + \begin{bmatrix} \mathbf{E}_1 & \mathbf{E}_2 \end{bmatrix}\begin{bmatrix} \tilde{\mathbf{B}}_{1,1} & 0 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} \mathbf{Z}_1 \\ \mathbf{Z}_2 \end{bmatrix}\Bigg) \qquad (36) \\
= {} & \eta\mathbf{X}_1\tilde{\mathbf{B}}_{1,1}\mathbf{Z}_1 + \eta g(\mathbf{X}_2\mathbf{H}_{2,1} + \mathbf{E}_1\tilde{\mathbf{B}}_{1,1})\mathbf{Z}_1 + \eta g^2(\ldots) \\
= {} & \eta\left(\mathbf{X}_1\tilde{\mathbf{B}}_{1,1} + g(\mathbf{X}_2\mathbf{H}_{2,1} + \mathbf{E}_1\tilde{\mathbf{B}}_{1,1})\right)\mathbf{Z}_1 \pmod{\mathcal{J}}.
\end{aligned}$$

The rest of the analysis is analogous. The rank of $\mathbf{A}$ modulo $\mathcal{J}$ distinguishes $\mathcal{B}$ and $\mathcal{B}'$.

4 Cryptanalysis of the GGH15-based candidate

4.1 The GGH15 Encoding Scheme

We use here notations similar to [29] for "GGH15 with safeguards". The encoding scheme from [26] is parametrized by a directed graph $G = (V, E)$ (with a single sink) and some integer parameters k, n, r, q (with $r \gg k$). Its plaintext space are matrices $\mathbf{S} \in R^{k\times k}$ (whose entries must be much smaller than q), and the encodings themselves are matrices $\mathbf{D} \in (R_n/qR_n)^{r\times r}$, and both plaintext and encoding matrices are associated with edges (or paths) in the graph.

For each vertex u in the graph we choose a random matrix $\mathbf{A}_u \in R_n^{k\times r}$ together with some trapdoor information τ_u [1,28,33], and another random invertible matrix $\mathbf{P}_u \in (R_n/qR_n)^{r\times r}$. For the source s and sink t we choose random small "bookend vectors" $\mathbf{J}_s$ and $\mathbf{L}_t$ and publish the two transformed vectors $\tilde{\mathbf{J}}_s := \mathbf{J}_s \cdot \mathbf{A}_s \cdot \mathbf{P}_s^{-1} \pmod q$ and $\tilde{\mathbf{L}}_t := \mathbf{P}_t \cdot \mathbf{L}_t$, to be used for zero-testing.

To encode a matrix $\mathbf{S} \in R^{k\times k}$ w.r.t. a path $(u \rightsquigarrow v)$, sample a low-norm error matrix $\mathbf{E} \in R_n^{k\times r}$, use the trapdoor τ_u to sample a small solution $\mathbf{D}$ to $\mathbf{A}_u\mathbf{D} = \mathbf{S}\mathbf{A}_v + \mathbf{E} \pmod q$, and finally output the encoding matrix $\mathbf{C} := \mathbf{P}_u\mathbf{D}\mathbf{P}_v^{-1} \bmod q$.

This scheme supports adding encoded matrices relative to the same path, and multiplying matrices relative to consecutive paths (with the result being defined relative to the concatenation of the two paths). The encoding invariant is that an encoding $\mathbf{C}$ of plaintext matrix $\mathbf{S}$ relative to the path $u \rightsquigarrow v$ satisfies $\mathbf{A}_u \cdot (\mathbf{P}_u^{-1}\mathbf{C}\mathbf{P}_v) = \mathbf{S}\mathbf{A}_v + \mathbf{E} \pmod q$ where $\mathbf{S}, \mathbf{E}$ and $\mathbf{D} := \mathbf{P}_u^{-1}\mathbf{C}\mathbf{P}_v \bmod q$ all have norm much smaller than q. The encoding scheme also supports a zero-test of encoding $\mathbf{C}$ relative to a path $s \rightsquigarrow t$, by checking that $\tilde{\mathbf{J}}_s\mathbf{C}\tilde{\mathbf{L}}_t$ is small, which holds when $\mathbf{S} = 0$ since $\tilde{\mathbf{J}}_s\mathbf{C}\tilde{\mathbf{L}}_t = \mathbf{J}_s\mathbf{A}_s\mathbf{P}_s^{-1} \cdot \mathbf{C} \cdot \mathbf{P}_t\mathbf{L}_t = \mathbf{J}_s(\mathbf{S}\mathbf{A}_s + \mathbf{E})\mathbf{L}_t = \mathbf{J}_s\mathbf{E}\mathbf{L}_t \pmod q$.

Consider two consecutive paths $s \rightsquigarrow u$ and $u \rightsquigarrow t$ and two encoding matrices $\mathbf{C}_1, \mathbf{C}_2$, encoding $\mathbf{S}_1, \mathbf{S}_2$ relative to these two paths, respectively. Then $\mathbf{C}_1\mathbf{C}_2$ is an encoding of $\mathbf{S}_1, \mathbf{S}_2$ relative to $s \rightsquigarrow t$, which means that $\mathbf{A}_s \cdot (\mathbf{P}_s^{-1}\mathbf{C}_1\mathbf{C}_2\mathbf{P}_t) = \mathbf{S}_1\mathbf{S}_2\mathbf{A}_t + \mathbf{E}' \pmod{q}$, but we can say more about the structure of the resulting noise $\mathbf{E}'$. Specifically, it is not hard to verify that (after zero-testing) we have

$$\begin{aligned}\tilde{\mathbf{J}}_s\mathbf{C}_1\mathbf{C}_2\tilde{\mathbf{L}}_t &= \mathbf{J}_s(\mathbf{S}_1\mathbf{S}_2\mathbf{A}_t + \underbrace{\mathbf{S}_1\mathbf{E}_2 + \mathbf{E}_1\mathbf{D}_2}_{\mathbf{E}'})\mathbf{L}_t \\ &= \mathbf{J}_s \cdot [\mathbf{S}_1|\mathbf{E}_1] \begin{bmatrix}\mathbf{S}_2\mathbf{A}_t + \mathbf{E}_2 \\ \mathbf{D}_2\end{bmatrix} \cdot \mathbf{L}_t \pmod{q},\end{aligned} \tag{37}$$

where $\mathbf{J}_s, \mathbf{L}_t$ are the bookend vectors, $\mathbf{E}_1, \mathbf{E}_2$ are the error matrices corresponding to the encoding $\mathbf{C}_1, \mathbf{C}_2$ respectively, and $\mathbf{D}_2 = \mathbf{P}_u^{-1}\mathbf{C}_2\mathbf{P}_t$ (all of which have low norm). Similarly, if we have three intervals $s \rightsquigarrow u$, $u \rightsquigarrow v$ and $u \rightsquigarrow t$ and three encoding matrices $\mathbf{C}_1, \mathbf{C}_2, \mathbf{C}_3$ for $\mathbf{S}_1, \mathbf{S}_2, \mathbf{S}_3$ relative to these paths, respectively, then

$$\tilde{\mathbf{J}}_s\mathbf{C}_1\mathbf{C}_2\mathbf{C}_3\tilde{\mathbf{L}}_t = \mathbf{J}_s \cdot [\mathbf{S}_1, \mathbf{E}_1] \begin{bmatrix}\mathbf{S}_2, & \mathbf{E}_2 \\ 0, & \mathbf{D}_2\end{bmatrix} \begin{bmatrix}\mathbf{S}_3\mathbf{A}_t + \mathbf{E}_3 \\ \mathbf{D}_3\end{bmatrix} \cdot \mathbf{L}_t \pmod{q}. \tag{38}$$

The GGHRSW obfuscator over GGH15 When using the GGH15 encoding scheme in the context of the GGHRSW obfuscator, we use a simple graph with two parallel chains leading to a sink. One minor technical issue to reconcile is that the plaintext space of GGH15 consists of only $k \times k$ matrices, while the GGHRSW construction needs to also encode the bookend vectors $\mathbf{J}, \mathbf{L}, \mathbf{J}', \mathbf{L}'$. This is best handled by combining these bookends with the GGH15 bookends $\mathbf{J}_s, \mathbf{L}_t$ from above. Namely we choose the bookends as matrices rather than vectors (but still keep the same structure for the rows/columns of these matrices), and then these matrices will be multiplied by the GGH15 bookends $\mathbf{J}_s, \mathbf{L}_t$ during zero-test, resulting in vectors $\mathbf{J}, \mathbf{L}, \mathbf{J}', \mathbf{L}'$ with the same structure as in Eqn. (2).

Another technical issue is that the GGH15 plaintext matrices must be small, whereas the GGHRSW construction requires that we multiply the plaintext matrices by the Kilian randomization matrices $\mathbf{K}, \mathbf{K}^{-1}$. Gentry et al. describe in [26, Section 5.2.1] a method for choosing "random matrices" where both $\mathbf{K}, \mathbf{K}^{-1}$ are small, but in fact a closer look at the error terms that we get reveals that the construction will still work even if only $\mathbf{K}^{-1}$ was small but $\mathbf{K}$ was not (as long as as we set $\mathbf{K}_0 = \mathbf{I}$). We stress that the structure of $\mathbf{K}$ plays no role in our attacks, so in the rest of the manuscript we ignore this issue.

4.2 Overview of our Attacks on the GGH15-based obfuscator

The main ingredient in our attack on the GGH15-based branching-program obfuscator is a method to recover some information about the scalars $\alpha_{i,b}$ (and $\alpha'_{i,b}$) that are used in this construction. Specifically, we use a zeroing technique adapted from the work of Coron, Lee, Lepoint and Tibouchi [17], to recover the ratios of (the products of) these $\alpha_{i,b}$'s for some equivalent subbranches, as we describe in Section 4.3 below. (Setting up the CLLT-style system of equations relies on the input-partitioning feature of underlying branching program.)

This step is completely algebraic, and hence the ratios that we recover do not give us a small representation of these scalars. Namely, while we learn the ratio β/γ for some small β, γ (each of them is a product of some α's), we do not recover the small β, γ themselves.

One way to mount a full attack to the obfuscator is to directly use factoring and principle-ideal-problem solvers to recover the $\alpha_{i,b}$'s from the known ratio β/γ. Once the bundling scalars $\alpha_{i,b}$ are known, we can mount an input-mixing attack to break the obfuscation. This yields classical sub-exponential time or quantum polynomial time attacks.

4.3 Step I: recovering ratios of the bundling scalars

Step I.1: Accumulating CLLT-style equations. Let $\mathcal{X}||\mathcal{Z} = [h]$ be a 2-partition of the branching program steps. Below we use honest evaluation of the branching program on many inputs of the form $u^{(i,j)} = x^{(i)}z^{(j)}$, where all the bits that only affect steps in $\mathcal{X}$ are in the $x^{(i)}$ part, all the bits that only affect steps in $\mathcal{Z}$ are in the $z^{(j)}$ part, and all the other bits are fixed. This notation *does not mean* that all the bits of $x^{(i)}$ must come before all the bits of $z^{(j)}$ in $u^{(i,j)}$, but it does mean that $u^{(i,j)}$ and $u^{(i,j')}$ can only differ in bits that affect steps in $\mathcal{Z}$, and similarly $u^{(i,j)}$ and $u^{(i',j)}$ can only differ in bits that affect steps in $\mathcal{X}$.

For such an input $u = xz$, we denote by $\mathbf{S}_x$ the plaintext product matrix of functional branch in the the $\mathcal{X}$ interval and by $\mathbf{S}_z$ the plaintext product matrix of the functional branch in the $\mathcal{Z}$ interval (including the bookends), and similarly for encodings $\mathbf{C}_x, \mathbf{C}_z$ and for the dummy branch. That is, we denote

$$\begin{aligned}
\mathbf{S}_x &:= \ \mathbf{J} \cdot (\textstyle\prod_{i \in \mathcal{X}} \mathbf{S}_{i,u_{\iota(i)}}), \ \mathbf{S}_z := \ (\displaystyle\prod_{i \in \mathcal{Z}} \mathbf{S}_{i,u_{\iota(i)}}) \cdot \mathbf{L}, \\
\mathbf{S'}_x &:= \mathbf{J}' \cdot (\textstyle\prod_{i \in \mathcal{X}} \mathbf{S'}_{i,u_{\iota(i)}}), \ \mathbf{S'}_z := (\displaystyle\prod_{i \in \mathcal{Z}} \mathbf{S'}_{i,u_{\iota(i)}}) \cdot \mathbf{L}', \\
\mathbf{C}_x &:= \ \tilde{\mathbf{J}} \cdot (\textstyle\prod_{i \in \mathcal{X}} \mathbf{C}_{i,u_{\iota(i)}}), \ \mathbf{C}_z := \ (\displaystyle\prod_{i \in \mathcal{Z}} \mathbf{C}_{i,u_{\iota(i)}}) \cdot \tilde{\mathbf{L}}, \\
\mathbf{C'}_x &:= \tilde{\mathbf{J}'} \cdot (\textstyle\prod_{i \in \mathcal{X}} \mathbf{C'}_{i,u_{\iota(i)}}), \ \mathbf{C'}_z := (\displaystyle\prod_{i \in \mathcal{Z}} \mathbf{C'}_{i,u_{\iota(i)}}) \cdot \tilde{\mathbf{L}'}
\end{aligned}$$

with the encoding arithmetic modulo q. We also denote by $\mathbf{E}_x, \mathbf{E}_z, \mathbf{E'}_x, \mathbf{E'}_z$ the error matrices in $\mathbf{C}_x, \mathbf{C}_z, \mathbf{C'}_x, \mathbf{C'}_z$ and

$$\mathbf{D}_x := \mathbf{C}_x \mathbf{P}_v, \ \mathbf{D'}_x := \mathbf{C'}_x \mathbf{P}_{v'}, \ \mathbf{D}_z := \mathbf{P}_v^{-1} \mathbf{C}_z \text{ and } \mathbf{D'}_z := \mathbf{P}_{v'}^{-1} \mathbf{C'}_z$$

(where v, v' are the vertices between $\mathcal{X}, \mathcal{Z}$ on the functional and dummy branches).

Following Eqn. (37) above, the honest evaluation of branching program on input $u = xz$ yields the element

$$\mathbf{w} \ := \ \mathbf{C}_x \mathbf{C}_z - \mathbf{C'}_x \mathbf{C'}_z \ = \ [\mathbf{S}_x, \mathbf{E}_x, -\mathbf{S}'_x, -\mathbf{E}'_x] \begin{bmatrix} \mathbf{S}_z \mathbf{A}_t + \mathbf{E}_z \\ \mathbf{D}_z \\ \mathbf{S}'_z \mathbf{A}_t + \mathbf{E}'_z \\ \mathbf{D}'_z \end{bmatrix} \pmod{q}. \quad (39)$$

If $u = xz$ is a zero of the function, then by construction we have $\mathbf{S}_x\mathbf{S}_z = \mathbf{S}'_x\mathbf{S}'_z = \beta\mathbf{J}\mathbf{L}$ for some scalar β, and in this case Eqn. (39) holds over the base ring R_n, not just modulo q.

We begin the attack by collecting many instances of Eqn. (39) for many $x^{(i)}$'s and $z^{(j)}$'s for which $u^{(i,j)} = x^{(i)}z^{(j)}$ is a zero, and put the corresponding w elements in a matrix. This yields the matrix equation:

$$\mathbf{W} := \mathbf{X}\mathbf{Z} := \begin{bmatrix} \mathbf{S}_{x^{(1)}}, & \mathbf{E}_{x^{(1)}}, & -\mathbf{S}'_{x^{(1)}}, & -\mathbf{E}'_{x^{(1)}} \\ \ldots, & \ldots, & \ldots, & \ldots \\ \mathbf{S}_{x^{(i)}}, & \mathbf{E}_{x^{(i)}}, & -\mathbf{S}'_{x^{(i)}}, & -\mathbf{E}'_{x^{(i)}} \\ \ldots, & \ldots, & \ldots, & \ldots \\ \mathbf{S}_{x^{(k)}}, & \mathbf{E}_{x^{(k)}}, & -\mathbf{S}'_{x^{(k)}}, & -\mathbf{E}'_{x^{(k)}} \end{bmatrix} \begin{bmatrix} \ldots, & \mathbf{S}_{z^{(j)}}\mathbf{A}_t + \mathbf{E}_{z^{(j)}}, & \ldots \\ \ldots, & \mathbf{D}_{z^{(j)}}, & \ldots \\ \ldots, & \mathbf{S}'_{z^{(j)}}\mathbf{A}_t + \mathbf{E}'_{z^{(j)}}, & \ldots \\ \ldots, & \mathbf{D}'_{z^{(j)}}, & \ldots \end{bmatrix}. \tag{40}$$

Since all the inputs are zeros of the function, then Eqn. (40) holds not only modulo q but also over the base ring R_n. As discussed in [29, Section 5.2], the matrix $\mathbf{Z}$ is inherently non-full-rank when considered modulo q, but will have full rank over R_n with high probability (since the $\mathbf{E}_z$'s are random and the $\mathbf{D}_z$'s are chosen at random in some cosets after the $\mathbf{E}_z$'s are fixed). Taking sufficiently many $z^{(j)}$'s, we can therefore ensure that the left kernel of $\mathbf{Z}$ is trivial, i.e., consisting of only the all-zero vector.[6] On the other hand, by using enough $x^{(i)}$'s we can ensure that the left-kernel of $\mathbf{W}$ (and therefore of $\mathbf{X}$) is non-trivial. The thrust of our attack will consist of collecting many vectors in this left-kernel, and using them to recover information about (ratios of) the $\alpha_{i,b}$'s.

Step I.2: Computing the left-kernel of $\mathbf{W}$. The CLLT-type attack computes the left-kernel (abbreviated as kernel in the rest of this paper) of $\mathbf{W}$, i.e. vectors $\mathbf{p}$ over R_n s.t. $\mathbf{pW} = \mathbf{0}$. Since $\mathbf{Z}$ has full rank, then such vector $\mathbf{p}$ must also be in the kernel of $\mathbf{X}$, so it is orthogonal to all its columns. In our attack we only use the fact that these vectors $\mathbf{p}$ are orthogonal to the $\mathbf{S}$'s parts of $\mathbf{X}$, namely we denote

$$\mathbf{Q} := \begin{bmatrix} \mathbf{S}_{x^{(1)}}, & -{\mathbf{S}'}_{x^{(1)}} \\ \ldots, & \ldots \\ \mathbf{S}_{x^{(i)}}, & -{\mathbf{S}'}_{x^{(i)}} \\ \ldots, & \ldots \\ \mathbf{S}_{x^{(k)}}, & -\mathbf{S}'_{x^{(k)}} \end{bmatrix} \tag{41}$$

and use the fact that every vector in the kernel of $\mathbf{X}$ must be in particular also in the kernel of $\mathbf{Q}$. We next recall the structure of $\mathbf{S}_{x^{(i)}}, {\mathbf{S}'}_{x^{(i)}}$ from Eqn. (3), namely we have

$$\begin{aligned} \mathbf{S}_{x^{(i)}} &= \alpha_{x^{(i)}}\mathbf{J} \times \mathsf{diag}(u_{x^{(i)}}, v_{x^{(i)}}, \mathbf{B}_{x^{(i)}}) \times \mathbf{K}_z; \\ \mathbf{S}'_{x^{(i)}} &= \alpha'_{x^{(i)}}\mathbf{J}' \times \mathsf{diag}(u'_{x^{(i)}}, v'_{x^{(i)}}, \mathbf{I}) \times \mathbf{K}'_z \end{aligned} \tag{42}$$

where $\mathbf{B}_{x^{(i)}}$ is the product of the branching-program matrices $\mathbf{B}_{i,b}$ over the interval $\mathcal{X}$, $u_{x^{(i)}}, v_{x^{(i)}}, u'_{x^{(i)}}, v'_{x^{(i)}}$ are the random diagonal entries, $\alpha_{x^{(i)}}, \alpha'_{x^{(i)}}$ are

[6] With typical parameters it is sufficient to use only four different $z^{(j)}$'s for that purpose.

the products of the α's on both the functional and dummy branches, and $\mathbf{K}_z, \mathbf{K'}_z$ are the Kilian randomization matrix at the beginning of the $\mathcal{Z}$ interval on the two branches.

Importantly, since all the $x^{(i)}z^{(j)}$'s are zeros of the function, then by Lemma 1 all the $\mathbf{B}_{x^{(i)}}$'s must be equal. We denote that matrix simply by $\mathbf{B}$, namely we have $\mathbf{S}_{x^{(i)}} = \alpha_{x^{(i)}}\mathbf{J} \times \mathsf{diag}(u_{x^{(i)}}, v_{x^{(i)}}, \mathbf{B}) \times \mathbf{K}_z$ for all i. Moreover, all the ratios of $\alpha_{x^{(i)}}/\alpha'_{x^{(i)}}$, $i \in [k]$ must also be equal due to Lemma 2, and below we denote that ratio by δ.

We can therefore re-write Eqn. (5) as follows:

$$\begin{aligned}\forall i \in [k],\ \ & [\mathbf{S}_{x^{(i)}}, -\mathbf{S}'_{x^{(i)}}] \\ &= \alpha_{x^{(i)}}[\mathbf{J} \times \mathsf{diag}(u_{x^{(i)}}, v_{x^{(i)}}, \mathbf{B}) \times \mathbf{K}_z, -\delta\mathbf{J}' \times \mathsf{diag}(u'_{x^{(i)}}, v'_{x^{(i)}}, \mathbf{I}) \times \mathbf{K}'_z] \\ &= \alpha_{x^{(i)}} \cdot [\ \underbrace{0,\ \tilde{v}_{x^{(i)}},\ \mathbf{b}}_{\mathbf{J}\times\mathsf{diag}(u_{x^{(i)}}, v_{x^{(i)}}, \mathbf{B})}\ ,\ \underbrace{0,\ \tilde{v'}_{x^{(i)}},\ \mathbf{b'}}_{\mathbf{J'}\times\mathsf{diag}(u'_{x^{(i)}}, v'_{x^{(i)}}, \mathbf{I})}\ ] \times \underbrace{\tilde{K}_z}_{\text{an invertible matrix}}\end{aligned} \tag{43}$$

(recall that by design, the first columns of $\mathbf{J}$ and $\mathbf{J}'$ are zero to erase the u and u' terms on the diagonal). The only $x^{(i)}$-sensitive terms are $\alpha_{x^{(i)}}$, $\tilde{v}_{x^{(i)}}$, and $\tilde{v'}_{x^{(i)}}$, and thus the rank of Q is exactly 3. The Kernel of $\mathbf{Q}$ therefore has dimension $k-3$, and it is contained in the ($(k-1)$-dimensional) space spanned by the vectors $[\alpha_{x^{(2)}}, -\alpha_{x^{(1)}}, 0, \ldots, 0]$, $[\alpha_{x^{(3)}}, 0, -\alpha_{x^{(1)}}, \ldots, 0]$, $\ldots$ $[\alpha_{x^{(k)}}, 0, 0, \ldots, -\alpha_{x^{(1)}}]$. In other words, every vector $\mathbf{p} = [p_1, p_2, \ldots, p_k]$ in this kernel must in particular satisfy the condition $\sum_i p_i \alpha_{x^{(i)}} = 0$.

Of course, the kernel of $\mathbf{X}$ (which is the linear space that our attack can recover) is only a subspace of the kernel of $\mathbf{Q}$, and hence it has an even lower dimension. However, the difference in dimension between $\mathsf{kernel}(\mathbf{Q})$ and $\mathsf{kernel}(\mathbf{X})$ is bounded by the dimensions of the error matrices $\mathbf{E}_{x^{(i)}}, \mathbf{E'}_{x^{(i)}}$, which is independent of the number of $x^{(i)}$'s. Namely, the number of columns in $\mathbf{E}_{x^{(i)}}, \mathbf{E'}_{x^{(i)}}$ together is only $2r$, hence the dimension of $\mathsf{kernel}(\mathbf{X})$ is at least $\mathsf{dim}(\mathsf{kernel}(\mathbf{Q})) - 2r = k - 3 - 2r$, where k is the number of $x^{(i)}$'s. If we have enough zeros of the branching-program, then we can take k to be much much larger than $2r + 3$.

Step I.3: Extracting the ratios. The kernel of $\mathbf{W}$ (or equivalently of $\mathbf{X}$) is a subspace of dimension at least $k - (2r + 3)$, all of which is orthogonal to the vector of $\alpha_{x^{(i)}}$'s. However, we do not have enough equations to recover the $\alpha_{x^{(i)}}$'s themselves, since there are k of them and we only have $k - (2r + 3)$ equations. Here we take advantage of the fact that the $\alpha_{x^{(i)}}$'s are not really k independent variables, rather each $\alpha_{x^{(i)}}$'s is obtained as a subset product of the $\alpha_{i,b}$'s that are used in the construction.

Specifically, let $J_x \subset [\ell]$ be set of input bits that only affect steps of the branching program in the interval $\mathcal{X}$, and for any just input bit $j \in J_x$ let us denote $\beta_{j,0} = \prod_{\iota(i')=j} \alpha_{i',0}$ and $\beta_{j,1} = \prod_{\iota(i')=j} \alpha_{i',1}$. Also, recalling that all the input bits outside J_x are fixed, we denote by β_0 the product of the $\alpha_{i',b}$ scalars that are used in all the steps that are not controlled by bits in J_x. Then every

$\alpha_{x^{(i)}}$ can be written as a subset product

$$\alpha_{x^{(i)}} = \beta_0 \cdot \prod_{j,b} {\beta_{j,b}}^{e(i,j,b)}$$

where the exponents $e(i,j,b)$ are all in $\{0,1\}$. This implies in particular that the number of $\alpha_{x^{(i)}}$ is at most $2^{2|J_x|}$.

Consider now what happens if we take all the products of two equations from the kernel. This will give us a set of at least $(k-2r-3)^2$ equations in the product variables $\gamma_{i_1,i_2} = \alpha_{x^{(i_1)}} \cdot \alpha_{x^{(i_2)}}$. But the γ_{i_1,i_2} are perhaps not all distinct: each of them can be written as a product $\gamma_{i_1,i_2} = \beta_0^2 \cdot \prod_{j,b} {\beta_{j,b}}^{e(i_1,i_2,j,b)}$ with the exponents in $\{0,1,2\}$, so the total number of distinct γ_{i_1,i_2} is at most $3^{2|J_x|}$ (which is smaller than $(2^{2|J_x|})^2$).

More generally, we can take products of upto c of our equations, and this will give us at least $(k-2r-3)^c$ equations, but the number of variables will still be upper-bounded by $(c+1)^{2|J_x|}$. If for some constant c we get $(k-2r-3)^c > (c+1)^{2|J_x|}$, then we have more equations than variables (which heuristically should still be linearly independent[7]) and we can solve the system and recover all the products $\gamma_{i_1,i_2,\ldots,i_c}$.

For example, in the extreme case where *every setting* of the input bits in J_x yields a zero of the function, we can collect as many as $k = 2^{|J_x|}$ equations from the kernel. If in addition $|J_x| > 1 + \log(2r+3)$ then $k > 2 \cdot (2r+3)$ and therefore $k - 2r - 3 > k/2 = 2^{|J_x|-1}$. In this case taking $c = 7$ is sufficient to get more equations than variables, since

$$(k-2r-3)^c = (k-2r-3)^7 > 2^{7(|J_x|-1)} > 2^{6|J_x|} = 8^{2|J_x|} = (c+1)^{2|J_x|}$$

as needed. We note that in this extreme case, we can get more equations than variables already when multiplying pairs of equations (i.e. let $c = 2$) from the kernel if we are careful about which pairs to multiply.

Once we have all the γ's, we can divide them by each other to get ratios of smaller products of the $\beta_{j,b}$'s from above (which are in turn products of the $\alpha_{i',b}$'s from the construction). In particular we can get ratios of individual β's, of the form $\beta_{j,b}/\beta_{j',b'}$, but we cannot get any better granularity. In particular we cannot separate the different $\alpha_{i',b}$ that are multiplied to form the $\beta_{j,b}$'s.

4.4 Step II: attacking the obfuscator

If we have a quantum computer, or we are willing to run a classical subexponential-time attack, we can implement a factoring oracle and a principal-ideal-problem solver, using [30,38,9,10,20]. Together, these solvers make it possible to recover (some of) the small scalars $\alpha_{i,b}$, $\alpha_{i',b'}$ from the ratios $\beta_{j,b}/\beta_{j',b'}$. Once we have these $\alpha_{i,b}$'s, we can use them in mixed-input attacks on the obfuscator. Namely, in some steps that are controlled by the j'th input bit we take the 0 matrix,

[7] This heuristic argument is similar to the one used for the multi-variate Coppersmith attack [15].

and in some other steps we take the 1 matrix, and this lets us (at least) check if these two matrices are the same.

In GGH15 (following GGHRSW), the small bundling scalars $\alpha_{i,b}$ and $\alpha'_{i,b}$ s.t. $\prod \alpha_{i,b} = \prod \alpha'_{i,b}$ are chosen by first generating a set of random small ζ_k's, let each $\alpha_{i,b}$ be a product of one or two of these ζ_k's, so that each of the product $\prod \alpha_{i,b}, \prod \alpha'_{i,b}$ correspond to the same subset of ζ_k's. A factoring oracle will recover the ideals generated by all the ζ_k's that happen to be prime (which happens with noticeable probability). Then a PIP solver will find the small ζ_k's themselves.[8] As each $\alpha_{i,b}$ is a product of very few of the ζ_k's, we would get some of the $\alpha_{i,b}$'s by trying all singletons and pairs of the ζ_k's.

Acknowledgments

We thank Rex Fernando, Peter Rasmussen, Amit Sahai, and the anonymous reviewers for their comments on a previous version of this report. Y.C. is supported by NSF grants CNS1012798, CNS1012910, CNS1413920 and AF1218461. Part of the research conducted while visiting Tel Aviv University, IST Austria, and IBM Thomas J. Watson Research Center. C.G. and S.H. are supported by the Defense Advanced Research Projects Agency (DARPA) and Army Research Office(ARO) under Contract No. W911NF-15-C-0236.

References

1. Miklós Ajtai. Generating hard instances of the short basis problem. In *International Colloquium on Automata, Languages, and Programming*, pages 1–9. Springer, 1999.
2. Martin R. Albrecht, Shi Bai, and Léo Ducas. A subfield lattice attack on overstretched NTRU assumptions - cryptanalysis of some FHE and graded encoding schemes. In *CRYPTO (1)*, volume 9814 of *LNCS*, pages 153–178. Springer, 2016.
3. Prabhanjan Ananth, Divya Gupta, Yuval Ishai, and Amit Sahai. Optimizing obfuscation: Avoiding barrington's theorem. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 646–658. ACM, 2014.
4. Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. In *CRYPTO (1)*, volume 9215 of *LNCS*, pages 308–326. Springer, 2015.
5. Daniel Apon, Nico Döttling, Sanjam Garg, and Pratyay Mukherjee. Cryptanalysis of indistinguishability obfuscations of circuits over ggh13. Cryptology ePrint Archive, Report 2016/1003, 2016.
6. Benny Applebaum and Zvika Brakerski. Obfuscating circuits via composite-order graded encoding. In *Theory of Cryptography*, pages 528–556. Springer, 2015.
7. Saikrishna Badrinarayanan, Eric Miles, Amit Sahai, and Mark Zhandry. Post-zeroizing obfuscation: New mathematical tools, and the case of evasive circuits. In *EUROCRYPT (2)*, volume 9666 of *LNCS*, pages 764–791. Springer, 2016.

[8] While the PIP solver will succeed in finding the prime ζ_k's, it will likely fail when applied to factors of the non-prime ζ_k's (since those are unlikely to be principal and/or very small).

8. Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Protecting obfuscation against algebraic attacks. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT*, volume 8441 of *LNCS*, pages 221–238. Springer, 2014.
9. Jean-François Biasse and Claus Fieker. Subexponential class group and unit group computation in large degree number fields. *LMS Journal of Computation and Mathematics*, 17(A):385–403, 2014.
10. Jean-François Biasse and Fang Song. Efficient quantum algorithms for computing class groups and solving the principal ideal problem in arbitrary degree number fields. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 893–902. SIAM, 2016.
11. Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. In *FOCS*, pages 171–190. IEEE Computer Society, 2015.
12. Zvika Brakerski and Guy N. Rothblum. Virtual black-box obfuscation for all circuits via generic graded encoding. In *TCC*, volume 8349 of *LNCS*, pages 1–25. Springer, 2014.
13. Jung Hee Cheon, Kyoohyung Han, Changmin Lee, Hansol Ryu, and Damien Stehlé. Cryptanalysis of the multilinear map over the integers. In *EUROCRYPT (1)*, volume 9056 of *LNCS*, pages 3–12. Springer, 2015.
14. 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. To be appear at ANTS, 2016.
15. Don Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *J. Cryptology*, 10(4):233–260, 1997.
16. Jean-Sébastien Coron, Craig Gentry, Shai Halevi, Tancrede 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 *Advances in Cryptology–CRYPTO 2015*, pages 247–266. Springer, 2015.
17. Jean-Sébastien Coron, Moon Sung Lee, Tancrède Lepoint, and Mehdi Tibouchi. Cryptanalysis of GGH15 multilinear maps. In *CRYPTO (2)*, volume 9815 of *LNCS*, pages 607–628. Springer, 2016.
18. Jean-Sébastien Coron, Moon Sung Lee, Tancrède Lepoint, and Mehdi Tibouchi. Zeroizing attacks on indistinguishability obfuscation over clt13. To be appear at PKC 2017. Cryptology ePrint Archive, Report 2016/1011, 2016.
19. Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Practical multilinear maps over the integers. In Ran Canetti and Juan A. Garay, editors, *CRYPTO (1)*, volume 8042 of *LNCS*, pages 476–493. Springer, 2013.
20. Ronald Cramer, Léo Ducas, Chris Peikert, and Oded Regev. Recovering short generators of principal ideals in cyclotomic rings. In *EUROCRYPT (2)*, volume 9666 of *LNCS*, pages 559–585. Springer, 2016.
21. Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT*, volume 7881 of *LNCS*, pages 1–17. Springer, 2013.
22. Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, pages 40–49. IEEE Computer Society, 2013.
23. Sanjam Garg, Craig Gentry, Shai Halevi, and Mark Zhandry. Functional encryption without obfuscation. In *TCC (A2)*, volume 9563 of *LNCS*, pages 480–511. Springer, 2016.

24. Sanjam Garg, Eric Miles, Pratyay Mukherjee, Amit Sahai, Akshayaram Srinivasan, and Mark Zhandry. Secure obfuscation in a weak multilinear map model. In *TCC (B2)*, volume 9986 of *LNCS*, pages 241–268, 2016.
25. Sanjam Garg, Pratyay Mukherjee, and Akshayaram Srinivasan. Obfuscation without the vulnerabilities of multilinear maps. Technical report, 2016.
26. Craig Gentry, Sergey Gorbunov, and Shai Halevi. Graph-induced multilinear maps from lattices. In *Theory of Cryptography*, pages 498–527. Springer, 2015.
27. Craig Gentry, Allison Bishop Lewko, Amit Sahai, and Brent Waters. Indistinguishability obfuscation from the multilinear subgroup elimination assumption. In *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*, pages 151–170. IEEE, 2015.
28. Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, pages 197–206, 2008.
29. Shai Halevi. Graded encoding, variations on a scheme. Cryptology ePrint Archive, Report 2015/866, 2015.
30. Arjen K. Lenstra, Hendrik W. Lenstra, Mark S. Manasse, and John M. Pollard. The number field sieve. In *STOC*, pages 564–572. ACM, 1990.
31. Huijia Lin. Indistinguishability obfuscation from constant-degree graded encoding schemes. In *EUROCRYPT (1)*, volume 9665 of *LNCS*, pages 28–57. Springer, 2016.
32. Huijia Lin and Vinod Vaikuntanathan. Indistinguishability obfuscation from ddh-like assumptions on constant-degree graded encodings. In *FOCS*, pages 11–20. IEEE Computer Society, 2016.
33. Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *Advances in Cryptology–EUROCRYPT 2012*, pages 700–718. Springer, 2012.
34. Eric Miles, Amit Sahai, and Mor Weiss. Protecting obfuscation against arithmetic attacks. *IACR Cryptology ePrint Archive*, 2014:878, 2014.
35. Eric Miles, Amit Sahai, and Mark Zhandry. Annihilation attacks for multilinear maps: Cryptanalysis of indistinguishability obfuscation over GGH13. In *CRYPTO (2)*, volume 9815 of *LNCS*, pages 629–658. Springer, 2016.
36. Eric Miles, Amit Sahai, and Mark Zhandry. Secure obfuscation in a weak multilinear map model: A simple construction secure against all known attacks. *IACR Cryptology ePrint Archive*, 2016:588, 2016.
37. Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation from semantically-secure multilinear encodings. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO (1)*, volume 8616 of *LNCS*, pages 500–517. Springer, 2014.
38. Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.
39. Joe Zimmerman. How to obfuscate programs directly. In *Advances in Cryptology-EUROCRYPT 2015*, pages 439–467. Springer, 2015.