Notes On GGH13 Without The Presence Of Ideals

Martin R. Albrecht, Alex Davidson, and Enrique Larraia*
Email: [martin.albrecht,alex.davidson.2014]@rhul.ac.uk

Royal Holloway, University of London

Abstract. We investigate the merits of altering the Garg, Gentry and Halevi (GGH13) graded encoding scheme to remove the presence of the ideal $\langle g\rangle$. In particular, we show that we can alter the form of encodings so that effectively a new g_i is used for each source group $\mathbb{G}_i$, while retaining correctness. This would appear to prevent all known attacks on IO candidates instantiated using GGH13. However, when analysing security in a simplified branching program model, we present an IO distinguishing attack that does not use $\langle g\rangle$. This result opens a counterpoint with the work of Halevi (EPRINT 2015) which stated that the core computational hardness problem underpinning GGH13 is computing a basis of this ideal. Our attempts seem to suggest that there is a structural vulnerability in the way that GGH13 encodings are constructed that lies deeper than the presence of $\langle g\rangle$. Tangentially, we observe that our attack is prevented when considering all the added machinery of IO candidates.

1 Introduction

The work of Garg, Gentry and Halevi [GGH13a] initiated the study of candidate *multilinear maps* (MMAPs). In short, a multilinear map $e : \mathbb{G}_1 \times \cdots \times \mathbb{G}_\kappa \mapsto \mathbb{G}_T$ maps κ elements $g_i \in \mathbb{G}_i$ to a single target element $g_T \in \mathbb{G}_T$ in target group $\mathbb{G}_T$.[1] More accurately, [GGH13a] constructed a *graded encoding scheme* (GES), informally defining intermediate bilinear maps between the source groups and thus allowing group operations on intermediate 'levels'. The actual construction provides 'noisy' approximations to the functionality of MMAPs and subsequent candidates [CLT13, GGH15] follow in the same vein. In fact, the common interface that we assume of a GES is similar to that of a levelled FHE scheme except that decryption is replaced with a public 'zero-testing' procedure. This allows the evaluator to learn whether a particular computation over encodings is equal to zero or not provided that the result is encoded at the top level of a computation hierarchy (e.g. after κ multiplications).

The importance of these constructions for theoretical cryptography has solidified with applications including semantically-secure, order-revealing encryption [BLR+15], attribute-based encryption for circuits [GGH+13c] and low-overhead broadcast encryption [BWZ14] to name a few. Although, perhaps the most important application is that of constructing candidates for indistinguishability obfuscation (IO) [GGH+13b, BGK+14, GMM+16]. All-known constructions of IO obfuscators require usage of a GES or MMAP, and analyse security in generic graded encoding models.

Unfortunately, the three candidates of GES [GGH13a, CLT13, GGH15] (from now on denoted as GGH13, CLT13 and GGH15 respectively) have been shown to be vulnerable to a wide-range of attacks; e.g. 'zeroizing' attacks [HJ16, CLR15, CLLT16, CLT14, CLLT17], attacks on the 'overstretched' NTRU assumption [ABD16, CJL16, KF17] and algebraic dependency attacks [MSZ16a, ADGM16, CGH17]. Zeroizing attacks are largely avoided in the realm of IO since they rely on lower-level encodings of zero being made available [HJ16, CLR15, CLLT16] or highly structured branching program constructions [CLLT17]. Attacks on 'overstretched' NTRU assumptions only affect GGH13, but can be avoided by increasing parameters. Finally, algebraic dependency attacks affect specific 'BGK-style' IO candidates (e.g. [AGIS14, BGK+14, BMSZ15, MSW14, PST14]) instantiated with the GGH13 GES and

* MA and EL were supported by the EPSRC grant EP/L018543/1 "Multilinear Maps in Cryptography". AD was supported by the EPSRC and the UK Government as part of the Centre for Doctoral Training in Cyber Security at Royal Holloway, University of London (EP/K035584/1).

[1] Here we describe an asymmetric MMAP, we can equally describe a symmetric variant where $\mathbb{G}_1 = \cdots = \mathbb{G}_\kappa = \mathbb{G}$.

rely on circuits outputting a sufficient number of zeroes. In short, the presence of a common generator g in all GGH13 encodings allows an adversary in an IO security game to create a basis for the ideal $\langle g \rangle$. This basis is then used in a distinguishing attack on the obfuscated circuit.

There have been attempts to develop 'immunised' IO constructions such as [GMM+16] (a combination of original proposals [GMS16, MSZ16b]). These immunisations construct branching programs that make finding algebraic dependences on zero-tested encodings (as described by [MSZ16a, ADGM16]) much more difficult, and analyse security in a weakened graded encoding model.[2] However, the cryptanalysis of Chen et al. [CGH17] seems to offer attacks that are still effective, even for these immunisations.[3]

This work. Our analysis is motivated by the cryptanalysis of Chen et al. [CGH17] and its applicability against 'immunised' IO candidates. We focus on algebraic dependency (or annihilation) attacks on IO candidates instantiated with GGH13, and their propensity to find representatives of the ideal $\langle g \rangle$. We investigate the possibility of making changes at the GES level to avoid the attacks described previously, rather than using ad-hoc fixes to IO constructions. In particular, we derive a variant of GGH13 where the common generator g is removed such that encodings on different levels i are cosets of the form $\alpha + I_i$ for κ ideals I_i. Concretely, we replace the usage of the short g in GGH13 with larger elements β_i that also depends on the level that the encoding is associated with (See Section 4 for more details). The correctness of zero-testing is achieved since the magnitude of the result is completely determined by the presence of the β_i's — the number of which differ in zero and non-zero encodings. The result is a GES that has no structural ideals that can easily be computed by a PPT adversary.

At first sight, this immediately prevents all known attacks that require an annihilation phase. On the other hand, it should be noted that, similarly to GGH13, our variant is trivially susceptible to zeroizing attacks and thus is immediately short of providing full MMAP functionality. As such, our alteration would only be a plausible candidate in situations where *multilinear jigsaw puzzle* (MJP) functionality is sufficient (e.g. IO [GGH+13b], order-revealing encryption [BLR+15]).

However, we find that the GES that we have derived is still vulnerable, in a simplified IO security game, to a variation of the annihilation attack given in [MSZ16a]. While we remove the ability of an adversary to learn ideals from our MJP scheme, we detail an attack that side-steps these measures and distinguishes based on the magnitude of zero-tested encodings. We interpret this result as a counterpoint to the work of Halevi [Hal15] where it is stated that the core computational hardness problem underpinning GGH13 is to establish a basis of $\langle g \rangle$. Given the similarity between our encodings and those of GGH13, our attack seems to highlight a structural fault that is exploitable even if the ideal testing capability of adversaries is removed. However, we stress that the attack is only possible because of distribution of our elements β_i that we use to replace g.

Finally, while our attack works in a simplified branching program model, the added machinery used in candidate obfuscators (such as 'multiplicative bundling' scalars and Kilian randomisation) render the attack *apparently* useless. This applies even for BGK-style obfuscators and seems to imply that our GES could still be used as a valid alternative for instantiating obfuscation candidates — even those that are believed insecure for GGH13. This may indicate a logical separation between the structure of the GGH13 MJP scheme and the variant proposed here. Unfortunately, a BGK-style obfuscator based on our GES still comes with no security guarantees and merely appears to resist currently available attacks in the weakened graded encoding model.

[2] This model allows all the same operations as the generic graded encoding model along with an additional step where the adversary is allowed to submit certain polynomial evaluations on the results of zero-testing.

[3] They are thwarted only by the usage of dual-input branching programs which are external to the security model considered. In fact, they note that parts of their attack take place externally to the WGEM and thus suggest that the model is incomplete.

Layout. Section 2 details the notation that we will use and a recap of rings, multilinear jigsaw puzzles, branching programs and definitions on algebraic dependence. In Section 3 we provide a brief overview of the GGH13 GES and the algebraic dependence attacks mentioned previously. In Section 4 we describe the changes we can make to GGH13 to remove the dependency on the ideal $\langle g \rangle$. In Section 5 we provide an analysis of the security of this variant when applied in differing IO security settings. In Section 6 we provide a final discussion of our results and possible future avenues for research.

2 Background and preliminaries

2.1 Notation

We may denote sets of the form $\{1, \ldots, n\}$ by $[n]$. For matrices M, we refer to the entry in row i and column j as $M[i, j]$. For ring elements x we also use the square-bracket notation $[x]_{\mathcal{S}}$ to represent an encoding of x with respect to some index set $\mathcal{S}$. For an algorithm A, we use the notation $w \leftarrow A(x, y, z)$ to denote that A outputs w on inputs x, y, z. For a set $\mathcal{X}$ we use the notation $x \leftarrow_{\$} \mathcal{X}$ to indicate that x is sampled from $\mathcal{X}$ using the uniform distribution. For elements $y \in R_q$ for some polynomial ring R_q, when referring to the *'magnitude'* of y we will mean $\|y\|_\infty$. For some distribution Y, we write $\mathsf{poly}(Y^n)$ to denote sampling a degree n polynomial with coefficients sampled from Y

2.2 Rings

We will be working over rings $R \coloneqq \mathbb{Z}[x]/\langle \phi(x) \rangle$ and $R_q \coloneqq R/qR$ for some degree $n = n(\lambda)$ integer polynomial $\phi(x) \in \mathbb{Z}[x]$ and a prime integer $q = q(\lambda) \in \mathbb{Z}$ — notably R_q is isomorphic to the ring $\mathbb{Z}_q[x]/\langle \phi(x) \rangle$. We perform addition in these rings component-wise in the coefficients of the polynomial elements and multiplication is performed via polynomial multiplication modulo $\phi(x)$ and, if applicable, q. An element in R (respectively R_q) can be viewed as a degree $(n-1)$ polynomial over $\mathbb{Z}$ (respectively $\mathbb{Z}_q$). We can represent such an element using the vector of its n coefficients (where these will be in the range $\{-\lfloor q/2 \rfloor, \ldots, \lfloor q/2 \rfloor\}$ for elements in R_q). We work with the polynomial $\phi(x) = x^N + 1$ with N a power of two. In particular, $\mathbb{Z}[x]/\langle \phi(x) \rangle$ is isomorphic to the ring of integers of the $2N$-th cyclotomic field.

Canonical embeddings. Let ζ_m denote a primitive m-th root of unity. The m-th cyclotomic number field $Q = \mathbb{Q}(\zeta_m)$ is the field extension of $\mathbb{Q}$ obtained by adjoining ζ_m. Let n be the degree of K over $\mathbb{Q}$, then there are n embeddings σ_i of $K \to \mathbb{C}$. These n embeddings correspond precisely to evaluation in each of the n distinct roots α_i of $\phi(x)$. In our case, $\psi(x)$ has $2 \cdot s_2 = n$ complex conjugate roots. Order the roots such that $\overline{\alpha_k} = \alpha_{s_2+k}$ for $k = 1, \ldots, s_2$. The *canonical embedding* $\sigma : K \to \mathbb{C}^n$ is defined as

$$a \mapsto (\sigma_1(a), \ldots, \sigma_{s_s}(a), \overline{\sigma_1}(a), \ldots, \overline{\sigma_{s_2}}(a))\,.$$

The canonical embedding maps into a space $H \subset \mathbb{C}^n$ given by

$$H = \{(x_1, \ldots, x_n) \in \mathbb{C}^n : \overline{x_j} = x_{s_2+j}, \forall 1 \leq j \leq s_2\}$$

which is isomorphic to $\mathbb{R}^n$ and we can represent the coordinates of $\sigma(a)$ by a real vector [CIV16]

$$(\tilde{a}_1, \ldots, \tilde{a}_n) \propto (\Re(\sigma_1(a)), \ldots, \Re(\sigma_{s_2}(a)), \Im(\sigma_1(a)), \ldots, \Im(\sigma_{s_2}(a)))\,.$$

This naturally induces a geometry on K with ℓ_2-norm $\|\cdot\|_2$ and ℓ_∞-norm $\|\cdot\|_\infty$:

$$\|a\|_2 = \|\sigma(a)\|_2 = \left(\sum_{i=1}^{n} |\tilde{a}_i|^2\right)^{1/2} \text{ and}$$
$$\|a\|_\infty = \|\sigma(a)\|_\infty = \max_i |\tilde{a}_i|.$$

Bounded distributions. When sampling our encodings we are required to define a B-bounded distribution, where all elements sampled from this distribution have an l_∞ norm that is bounded by B. In this section we will formally define such a distribution.

Definition 1. *(B-bounded element) An element $p \in R$ is called* B-bounded *if* $\|p\|_\infty \leq B$.

Definition 2. *(B-bounded distribution) A distribution ensemble $\{\chi_\lambda\}_{\lambda\in\mathbb{N}}$, supported over R, is called* B-bounded *(for $B = B(\lambda)$) if for all p in the support of χ_λ, we have $\|p\|_\infty < B$. In other words, a B-bounded distribution over R outputs a B-bounded polynomial.*

Lemma 1. ([LTV12]) *Let $n \in \mathbb{N}$, let $\phi(x) = x^n + 1$ and let $R = \mathbb{Z}[x]/\langle\phi(x)\rangle$. For any $s, t \in R$,*

$$\|s \cdot t\| \leq \sqrt{n} \cdot \|s\| \cdot \|t\| \qquad \textit{and} \qquad \|s \cdot t\|_\infty \leq \|s\|_\infty \cdot \|t\|_\infty$$

Corollary 1. ([LTV12]) *Take $n, \phi(x), R$ as before. Let $s_1, \ldots, s_k \leftarrow_\$ \chi$ where χ is a B-bounded distribution over the ring R. Then*

$$s := \prod_{i=1}^{k} s_i$$

is $(n^{k-1}B^k)$-bounded.

Gaussian sampling. For any real $r > 0$ the *Gaussian function* on $\mathbb{R}^n$ centred at $\mathbf{c}$ with parameter r is defined as:

$$\forall \mathbf{x} \in \mathbb{R}^n \quad : \quad \rho_{r,c}(\mathbf{x}) := e^{-\pi||\mathbf{x}-\mathbf{c}||^2/r^2}$$

Definition 3. *For any $n \in N$ and for any $\mathbf{c} \in \mathbb{R}^n$ and real $r > 0$, the Discrete Gaussian distribution over $\mathbb{Z}^n$ with standard deviation r and centred at $\mathbf{c}$ is defined as:*

$$\forall \mathbf{x} \in \mathbb{Z} : D_{\mathbb{Z}^n,r,\mathbf{c}} := \frac{\rho_{r,\mathbf{c}}(\mathbf{x})}{\rho_{r,\mathbf{c}}(\mathbb{Z}^n)}$$

where $\rho_{r,\mathbf{c}}(\mathbb{Z}^n) := \sum_{\mathbf{x}\in\mathbb{Z}^n} \rho_{r,\mathbf{c}}(\mathbf{x})$ is a normalisation factor.

The work of [MR04] showed that the discrete Gaussian distribution over $\mathbb{Z}^n$ with standard deviation r outputs elements that are $(r\sqrt{n})$-bounded with high probability ($\geq 1 - 1/2^{-n+1}$). We can then define the *truncated* Gaussian distribution that is $(r\sqrt{n})$-bounded and is statistically close to the discrete Gaussian.

The truncated Gaussian with standard deviation r and centred at $\mathbf{c}$ will be denoted by $\bar{D}_{\mathbb{Z}^n,r,\mathbf{c}}$ and can be defined by sampling polynomials according to the discrete Gaussian ($D_{\mathbb{Z}^n,r,\mathbf{c}}$) and repeating any samples that are not $(r\sqrt{n})$-bounded. We note that this distribution is statistically close to $D_{\mathbb{Z}^n,r,\mathbf{c}}$ as shown in [LTV12]. For the case where $\mathbf{c} = 0$ we will simply write $\bar{D}_{\mathbb{Z}^n,r}$.

Our GES (Section 4) relies on distinguishing between products of $\kappa - 1$ and κ elements. For this, we sample real vectors $(\tilde{a}_1, \ldots, \tilde{a}_n)$ with each coordinate sampled from a Gaussian distribution conditioned on a minimum size through rejection sampling. Mapping these real vectors to elements in K produces the desired distribution in K. We then discretise, i.e. randomised round each coordinate to an integer to obtain elements in $\mathbb{Z}[x]/\langle\phi(x)\rangle$ as usual.

Thus, we may infer the magnitude of elements that are sampled from certain distributions and latterly what the magnitude of such an element is expected to be after multiplying any number of these elements is. We can use this information to make statements on the size of encodings that are made up of elements sampled from such B-bounded distributions.

2.3 Multilinear jigsaw puzzles

In the introduction we referred to IO candidates instantiated from graded encoding schemes. In fact, we instantiate IO from multilinear jigsaw puzzles (MJPs) — a restricted variant of a GES where lower-level encodings of zero are not permitted and only certain types of multilinear form can be computed. From now on we will use the following MJP formalisation when referring to the functionality required for constructing IO rather than the wider GES framework.

Definition 4 (MJP Scheme). *A multilinear jigsaw puzzle consists of two algorithms* $(\mathsf{JGen}, \mathsf{JVer})$ *that generate the puzzle and verify a solution to the puzzle, respectively. We explain the algorithms in detail.*

Puzzle generation: *Algorithm* JGen *comprises the triple of sub-algorithms* $(\mathsf{JInstGen}, \mathsf{JEnc}, \mathsf{JGenPuzz})$ *described as such:*

- $\mathsf{JInstGen}(1^\lambda, 1^\kappa)$: *On input the security parameter* λ *and multilinearity* κ*, this algorithm outputs a set of private parameters* sk *needed to encode ring elements, and a set of public parameters* $\mathsf{pp} = (\mathsf{prms}, \mathsf{evk}, \mathsf{ztk})$*. The last two components of the public tuple are necessary to perform algebraic operations over the encodings, and for zero-testing, respectively. The system-wide parameters* prms *include a prime* q *defining the working ring, a set universe* $\mathcal{U}$*, and a partition* $\{\mathcal{S}_1, \ldots, \mathcal{S}_\kappa\}$ *of* $\mathcal{U}$*.*
- $\mathsf{JEnc}(\mathsf{prms}, \mathsf{sk}, \mathcal{S}, a)$: *On input* sk*, a set* $\mathcal{S} \subset \mathcal{U}$ *and* $a \in \mathbb{Z}_q$*, this algorithm outputs an encoding* v *relative to the set* $\mathcal{S}$*.*
- $\mathsf{JGenPuzz}(1^\lambda, 1^\kappa, l, A)$: *Takes as input the security and multilinearity parameters,* $l \in \mathbb{N}$ *and a set* $A = (A_1, \ldots, A_l)$*, where* A_i *is a set of values* $\{a_j\}_{j \in [m_i]}$ *that will be encoded with respect to index set* $\mathcal{S}_i$*. First it runs* $\mathsf{JInstGen}(1^\lambda, 1^\kappa)$ *to receive system parameters* $(\mathsf{sk}, \mathsf{pp} = (\mathsf{prms}, \mathsf{evk}, \mathsf{ztk}))$*. It then runs* JEnc *on inputs* $(\mathsf{prms}, \mathsf{sk})$ *and each element* $(\mathcal{S}_i, a_j) \in A_i$ *to receive encodings* $(\mathcal{S}_i, v_j) \in C_i$*.*

Let $\mathsf{puzzle} = (C_1, \ldots, C_l)$ *and let* $X = ((\mathcal{S}_1, v_1), \ldots, (\mathcal{S}_l, v_l))$*, then we define* (X, puzzle) *as the output of* JGen *where* X *is kept secret and* puzzle *is the public output.*

Puzzle verification: *Algorithm* JVer *takes as input the public parameters* $\mathsf{pp} = (\mathsf{prms}, \mathsf{evk}, \mathsf{ztk})$*, the public output* puzzle *of* JGen *and some multilinear form* F *(the solution to the puzzle). It outputs either acceptance or rejection. More formally, following [MSZ16a], we split the verification into three sub-algorithms* $\mathsf{JVer} = (\mathsf{JCompute}, \mathsf{JZTParam}, \mathsf{JTest})$*. This helps in capturing the weakened grading encoding security model [MSZ16a, GMM+16].*

- $\mathsf{JCompute}(\mathsf{prms}, \mathsf{evk}, \mathsf{puzzle}, F)$: *On input the encodings in* puzzle *and some* valid *multilinear form* F*, this outputs the encoding* $(\mathcal{S}, v) = F(\mathsf{puzzle})$ *for* $\mathcal{S} \subseteq \mathcal{U}$*. We will sometimes abuse notation and simply write the output of the algorithm as* $F(\mathsf{puzzle})$*.*
- $\mathsf{JZTParam}(\mathsf{prms}, \mathsf{ztk}, (\mathcal{S}, v))$: *On input encoding* $(\mathcal{S}, v)$ *it first checks if* $\mathcal{S} = \mathcal{U}$ *and if not aborts. If true, it outputs the ring element* δ*. In an honest execution we have that* $(\mathcal{S}, v) \leftarrow \mathsf{JCompute}(\mathsf{puzzle}, F)$*.*
- $\mathsf{JTest}(\mathsf{prms}, \delta)$: *On input ring element* δ *it returns* 1 *or* 0*. In an honest execution we have that* $\delta \leftarrow \mathsf{JZTParam}(\mathsf{prms}, \mathsf{ztk}, (\mathcal{U}, v))$*.*

In the above definition, by *valid* multilinear form, we mean some sort of computation that respects the computation laws of a graded encoding scheme and outputs a top-level encoding [GGH+13b]. For instance, for any encodings $(\mathcal{S}_1, v_1), (\mathcal{S}_2, v_2)$ we have an addition operation that is defined when $\mathcal{S} = \mathcal{S}_1 = \mathcal{S}_2$ and outputs the encoding $(\mathcal{S}, v_1 + v_2)$. We also have multiplication that is defined when $\mathcal{S}_1 \cap \mathcal{S}_2 = \emptyset$ and results in an encoding $(\mathcal{S}, v_1 \cdot v_2)$ for $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2$. The output of these operations is said to be a top-level encoding when $\mathcal{S} = \mathcal{U}$.

Definition 5 (MJP Correctness). *A jigsaw verifier* JVer *is* correct *with respect to* $(\mathsf{pp}, (X, \mathsf{puzzle}), F)$ *if either* $F(X) = (\mathcal{U}, 0)$ *and* $\mathsf{JVer}(\mathsf{puzzle}, F) = 1$ *or* $F(X) \neq (\mathcal{U}, 0)$ *and* $\mathsf{JVer}(\mathsf{puzzle}, F) = 0$*. Otherwise it is* incorrect *on* F*.*

We specifically require that JVer is correct on all but negligibly many forms (see [GGH+13b] for an explanation of the requirement).

Security. Characterising the security that should be offered by a MJP is one of the difficulties confronted by constructions of IO. In short, constructions of IO are proven secure in a generic model where encodings are treated as random handles and all operations that can be performed are interacted with via oracle calls. Yet, as discussed above, current MJP constructions do not justify the use of such a model, i.e. they are broken by attacks which fall our side of this model. See [MSZ16a, GMM+16] for more details.

2.4 Branching programs

Let $L = L(\lambda)$, $\nu = \nu(\lambda)$ and $d = d(\lambda)$ be parameters dependent on the security parameter λ. Let $\mathsf{inp} : [L] \mapsto [\nu]^d$ be some 'input' function. Let $\{M_{(b_1,\ldots,b_d),l}\}$ be a set of matrices individually sampled from $\mathbb{Z}_q^{5\times 5}$ for $b_1, \ldots, b_d \in \{0,1\}$ and $l \in [L]$. Let $M_0 \in \mathbb{Z}_q^{5\times 1}$, $M_{L+1} \in \mathbb{Z}_q^{1\times 5}$ be two vectors, these are known as 'bookends' and are used for guaranteeing a single element output. Define

$$\mathcal{M} \coloneqq (L, \nu, d, \mathsf{inp}, \{M_{x_{\mathsf{inp}(l)},l}\}_{l\in[L]}, M_0, M_{L+1}) \tag{1}$$

to be a matrix branching program (MBP) of length L, input width ν and arity d. We can evaluate $\mathcal{M}$ on inputs $x \in 2^\nu$ where $x_s = x[s]$ and we denote such an evaluation by $\mathcal{M}(x)$.[4] The input function inp chooses the bits in the input x that are examined at each layer l of the branching program. Clearly $|\mathsf{inp}(i)| = d$ where $\mathsf{inp}(i)[y]$ is equal to the y^{th} component of $\mathsf{inp}(i)$. In total, we have that the branching program contains $2^d L + 2$ matrices. Let $x_{\mathsf{inp}(l)} = (x_{\mathsf{inp}(l)[1]}, \ldots, x_{\mathsf{inp}(l)[d]})$, we evaluate the branching program on an input x by computing

$$\mathcal{M}(x) \coloneqq M_0 \cdot \left(\prod_{l=1}^{L} M_{x_{\mathsf{inp}(l)},l}\right) \cdot M_{L+1}. \tag{2}$$

Using Barrington's theorem we can associate a circuit C with a branching program $\mathcal{M}_C$. We have that the branching program (without bookend vectors) evaluates to the identity matrix on an input x if and only if $C(x) = 0$. Since this theorem is commonly used in the construction of IO candidates this formulation of correctness applies to our situation. It is common to structure a branching program $\mathcal{M}_C$ such that $\mathcal{M}(x) = 0$ when $C(x) = 0$. To ensure this, we can construct a dummy branching program that contains only identity matrices with the same bookend vectors as in the functional branching program. We then compute the dummy branch on the same input as the functional branch and subtract the dummy output from the functional output. All current obfuscators only consider branching programs that take either single [GGH+13b] or dual [BGK+14] inputs i.e. cases where $d = 1$ or $d = 2$.

Definition 6. (Functional equivalence) *Let $\mathcal{X}$ be the set of valid inputs for two branching programs $\mathcal{M}_0, \mathcal{M}_1$ of length L, input length ν and arity d. We say that $\mathcal{M}_0, \mathcal{M}_1$ are* functionally equivalent *(or $\mathcal{M}_0 \equiv \mathcal{M}_1$) if, for any input $x \in \mathcal{X}$ then:*

$$\mathcal{M}_0(x) = 0 \quad \text{if and only if} \quad \mathcal{M}_1(x) = 0.$$

As above, we can alter the branching program computation to ensure that only a single value is output rather than matrices.

Remark 1. Note that we can pad the length of the branching program to any required length by simply appending the required number of identity matrices to the end of the branching program. These matrices clearly do not alter the result of the program evaluation.

[4] We use 5×5 matrices as these are sufficient for Barrington's theorem [Bar89].

2.5 IO from branching programs

The majority of current IO candidates make use of branching programs when constructing an obfuscated version of a circuit $C(\cdot)$. This generalised approach is developed from the randomised branching program model used by [MSZ16a] — using Barrington's theorem to convert a fan-in 2 circuit, C, of depth D into a branching program $\mathcal{M}$ of the form above with length $L = O(4^D)$. The construction we detail here is heavily generalised but follows the BGK-style obfuscation candidates of [AGIS14, BGK+14, BMSZ15, MSW14, PST14]. Obfuscators such as [GGH+13b, GMM+16] use more complicated randomisation procedures.

To obfuscate the program, we first apply Kilian's randomisation technique [Kil88] by randomly sampling invertible matrices $\mathcal{R}_0, \ldots, \mathcal{R}_{L+1}$, sampling $2L$ random non-zero scalars $\epsilon_{b,l} \leftarrow\!\!\mathrm{\$}\, \mathbb{Z}_q$ and then constructing randomised matrices

$$\widetilde{M_{b,l}} = \epsilon_{b,l} \cdot \mathcal{R}_{l-1}^{-1} M_{b,l} \mathcal{R}_l$$

along with bookend vectors

$$\widetilde{M_0} = \epsilon_0 \cdot M_0 \mathcal{R}_0, \quad \text{and} \quad \widetilde{M_{L+1}} = \epsilon_{L+1} \cdot \mathcal{R}_{L+1}^{-1} M_{L+1}.$$

Notice that the following holds:

$$\widetilde{M_0} \cdot \left(\prod_{l=1}^{L} \widetilde{M}_{x_{\mathsf{inp}(l)},l} \right) \cdot \widetilde{M_{L+1}} = \tilde{\epsilon} \cdot M_0 \cdot \left(\prod_{l=1}^{L} M_{x_{\mathsf{inp}(l)},l} \right) \cdot M_{L+1}$$

for the multiplicative bundling scalar $\tilde{\epsilon} = \epsilon_0 \epsilon_{L+1} \prod_{i=1}^{L} \epsilon_{x_{\mathsf{inp}(i)},i}$. This means that, if we replace the matrices in the branching program with the randomised matrices then we still compute the same function. These randomisations procedures help prevent partial evaluation and input mixing attacks on obfuscated programs. Similarly to the work of [BGK+14] we could make use of a straddling set structure when encoding elements to prevent further algebraic attacks. This is tangential to the security model that we consider and so we do not describe this procedure here.

Finally the entries of each matrix $\widetilde{M_{b,l}}$ are encoded using an MJP scheme with respect to a source index set $\mathcal{S}_{b,l}$. This allows the security of the construction to be analysed in the generic graded encoding model (to be consistent with previous work), essentially limiting an adversary to computing multilinear operations and zero-testing on top-level encodings. We denote the encoded matrices by $\widehat{M_{b,l}}$, the bookend vectors by $\widehat{M_0}, \widehat{M_{L+1}}$ and the obfuscated branching program by $\widehat{\mathcal{M}}$. The index sets are define such that the output of $\widehat{\mathcal{M}}$ is an encoding with respect to a top-level set $\mathcal{U}$.

Evaluation. When evaluating the branching program on an input x the bookend vectors and a dummy program execution ensure that a single element is propagated from the computation. Using the randomised branching program

$$\widehat{\mathcal{M}} := (L, \nu, d, \mathsf{inp}, \{\widehat{M}_{x_{\mathsf{inp}(l)},l}\}_{l \in [L]}, \widehat{M_0}, \widehat{M_{L+1}})$$

we learn a top-level encoded element where the encoded value is 0 if and only if the circuit that was obfuscated also evaluates to 0 on x.[5] That is, $\widehat{\mathcal{M}}_C(x) = [0]_{\mathcal{U}}$ iff $C(x) = 0$ and, since $[0]_{\mathcal{U}}$ is a top-level encoding, we can use the zero-test procedure to learn the output of the obfuscated circuit.

2.6 Algebraic dependence

Here, we list definitions and key results taken from the work of [Kay09] that we use in the security analysis of our MJP scheme. In short, we articulate the formalisation of expressing algebraic dependencies for a set of polynomials sampled from a particular field.

[5] The use of the bundling scalars ensure that any input x such that $C(x) = 1$ satisfies $\widehat{\mathcal{M}}_C(x) \neq [0]_{\mathcal{U}}$

Definition 7. *Let $f = (f_1, \ldots, f_k)$ be a vector of k polynomials of degree $\leq d$, where each $f_i \in \mathbb{F}[y_1, \ldots, y_n]$ is an n-variate polynomial over $\mathbb{F}$. A non-zero polynomial $A(t_1, \ldots, t_k) \in \mathbb{F}[t_1, \ldots, t_k]$ is said to be an* annihilating polynomial *for f if $A(f_1, \ldots, f_k) = 0$. The polynomials $f_1, \ldots, f_k$ are said to be* algebraically dependent *if such an annihilating polynomial exists.*

Definition 8. *Let $f = (f_1, \ldots, f_k)$ be a vector of k polynomials as above where f' represents some subset of algebraically independent polynomials of maximal size k (i.e. for any $f_{k+1} \in \mathbb{F}[y_1, \ldots, y_n]$ then the set $f' \cup f_{k+1}$ is algebraically dependent). Then the* algebraic rank *of the set of polynomials f is k.*

Theorem 1 (Theorem 2 [Kay09]). *Let $f_1, \ldots, f_k \in \mathbb{F}[x_1, \ldots, x_n]$ be a set of k polynomials in n variables over the field $\mathbb{F}$. Then this set of polynomials has algebraic rank k if and only if the Jacobian matrix, $Jf(x)$, has rank k.*

Corollary 2 ([Kay09, BS83]). *There exists a randomized polynomial time algorithm that on input a set of k arithmetic circuits over a field $\mathbb{F}$, determines if the polynomials computed by these arithmetic circuits are algebraically dependent or not.*

Remark 2. The algorithm mentioned by Corollary 2 essentially requires submitting random values in place of the variables in the Jacobian matrix $Jf(x)$. By the Schwarz-Zippel lemma, the rank of the symbolic matrix is likely to be the same as the rank of the matrix evaluated on random inputs with high probability. As such we can calculate the algebraic rank for a given system of polynomials.

3 GGH13 and annihilation attacks

3.1 GGH13 overview

The space for GGH13 encodings is $R_q = R/qR$ where q is some big integer and $R = \mathbb{Z}[x]/(x^m + 1)$ for $m \in \mathbb{N}$. The plaintext ring is defined by $R_g = R/gR$ where g is a small element in the ring. A GGH13 encoding takes the form $v = (\alpha + rg)/z \mod q$ where z is some uniformly random value — z and g are secret — α is the encoded plaintext value and r is some small random value, all these values are sampled from some error distribution, χ, over R_q.

The denominators z enforce the levels of the GES, where we can sample one global z for the symmetric case and $z_1, \ldots, z_\kappa$ in the asymmetric case, we will consider the asymmetric case unless otherwise stated. Where an encoding v has a denominator z_i we will say that v is encoded at level $\mathcal{S}_i$ where there are κ such index sets. Additions and multiplications are carried out by simply adding and multiplying encodings directly. Clearly, additions of encodings indexed at the same level results in another encoding at that level. Multiplying two encodings, indexed by z_1 and z_2 respectively, results in an encoding at level $\mathcal{S}_1 \cup \mathcal{S}_2$.

Finally, there is a public zero-test parameter

$$p_{zt} = \frac{h \cdot \prod_{i=1}^{\kappa} z_i}{g}$$

for some 'smallish' $h \in R_q$.[6] We can learn whether an encoding $(\mathcal{U}, v)$ (e.g. top-level with denominator $z_1 \cdot \cdots \cdot z_\kappa$) encodes zero or not by computing $p_{zt} \cdot v$ and seeing if the result is small.

The functionality described can be adapted to construct a correct MJP scheme [GGH+13b].

[6] The exact magnitude is not important for the attack in [MSZ16a] as long as $h \ll q$.

3.2 Annihilation attacks on GGH13

Let $\widehat{\mathcal{M}}$ be a randomised branching program that has entries encoded as GGH13 elements and each pair of matrices $\widehat{M}_{b,l}$ and bookends $\widehat{M}_0, \widehat{M}_{L+1}$ are encoded with respect to the levels $l \in \{0, \ldots, L+1\}$. Let $x \in \mathcal{X}$ be some valid input for $\widehat{\mathcal{M}}$ and let $\mu_x \leftarrow \widehat{\mathcal{M}}(x)$ be the output. Finally denote $\delta_x = p_{zt} \cdot \mu_x = \mathsf{JZTParam}(\mu_x)$ as the zero-tested output.

A top-level GGH13 encoding will have the following form:

$$\delta_x = \widetilde{\alpha}_x \cdot g^{-1} + \gamma_{1,x} + \gamma_{2,x} \cdot g + \ldots + \gamma_{\kappa,x} \cdot g^{\kappa-1} \tag{3}$$

after zero-testing has occurred. The target of the annihilation attacks is the polynomial $\gamma_{1,x}(\alpha, r)$ which is linear in the unknown sampled elements r_j from each encoding v_j. Using a change of variables in the branching program it is possible to assume that the adversary has knowledge of the values α_j that are encoded in each of the matrices [MSZ16a] (see Section 5.3 for more details). By choosing enough inputs x such that $\alpha_x = 0$, the adversary is able to guarantee that there exists an annihilating polynomial Q for the set of $\gamma_{1,x}$ polynomials. In fact, the work of [MSZ16a] explicitly gives a description of Q for a given single-input branching program $\mathcal{M}$.

Consequently, the result $\rho_x \leftarrow Q(\{\delta_x\}_x)$ results in some output where the $\gamma_{1,x}$ polynomials are eliminated. In particular, this means that $\rho_x \in \langle g \rangle$. By computing enough outputs, an adversary can heuristically construct a basis of $\langle g \rangle$. The attack concludes by specifying a functionally equivalent $\mathcal{M}'$ where the set of polynomials $\gamma'_{1,x}$ are not annihilated by Q. The work of [MSZ16a] show that it is possible to construct $\mathcal{M}, \mathcal{M}'$ such that a PPT adversary with obfuscated access to either of the circuits can first construct a basis of $\langle g \rangle$[7] and then secondly distinguish between the circuits. Distinguishing is possible since $\rho'_x \leftarrow \widehat{\mathcal{M}}'(x)$ is not in $\langle g \rangle$ and so they are able to distinguish using the basis computed in the first step.

4 GGH13 without ideals

The main component of this note is our analysis of the security of IO candidates when instantiated with a variant of GGH13 where ideals cannot be efficiently found. In the following, we give an overview of our scheme, in Appendix A, we give a formal MJP realisation.

4.1 Overview of encodings

Let $R = \mathbb{Z}[x]/\langle \phi(x) \rangle$ be a ring and $R_q = R/qR$ be the quotient ring for a large prime q, with an accompanying error distribution $\chi = \bar{D}_{\mathbb{Z}^n,\sigma}$ for parameter σ. The ring R_q will define the space of encodings and all operations take place there.

Let $\alpha \in R_q$ be some non-zero polynomial with small coefficients. Sample a polynomial $r \leftarrow_\$ \mathsf{poly}(\chi)$ with small coefficients and sample z_i uniformly from R_q for $1 \leq i \leq \kappa$. Finally, sample β_i such that $\sqrt[\kappa+1]{q} < \|\beta_i\|_\infty < \sqrt[\kappa]{q}$. We refer the reader to Section 2.2 for more details on how to sample the β_i.

A level 1 encoding of α with respect to some set $\mathcal{S}_i$ takes the form:

$$[v]_q = \frac{\alpha + r/\beta_i}{z_i} \mod q \tag{4}$$

where the values z_i, β_i enforce the leveled structure that we require from the specification of an MJP scheme. It is easy to see that β_i^{-1} corresponds to using a different g_i in GGH13. We reiterate that β_i has to be sampled in a different way (e.g. no longer as small elements) to ensure correctness for zero-testing.

[7] It is possible to find inputs for both circuits that allow the adversary to construct a basis of $\langle g \rangle$.

Remark 3. It is possible to sample higher-level encodings by encoding with respect to $\prod_{i\in\mathcal{S}} z_i$ and $\prod_{i\in\mathcal{S}} \beta_i$ for some index set $\mathcal{S}$. We do not require this functionality for a MJP scheme.

4.2 Operations

Let $\mathcal{U}$ refer to the top-level index where zero-testing can take place.

Addition of encodings. Let v_1, v_2 be encodings with respect to the same index set $\mathcal{S} \subseteq \mathcal{U}$. Then we can compute additions of these encodings by simply computing $v = v_1 + v_2$, the result is an encoding of the form

$$[v]_q = \frac{\alpha_1 + \alpha_2 + (r_1 + r_2)/\beta_{\mathcal{S}}}{z_{\mathcal{S}}} \mod q$$

Multiplication of encodings. Let v_1, v_2 be encodings with respective index sets $\mathcal{S}_i$ and $\mathcal{S}_j$ such that $\mathcal{S}_i \cup \mathcal{S}_j \subseteq \mathcal{U}$. A multiplication of these encodings is calculated by multiplying the encodings directly, i.e. $v = v_1 \cdot v_2$ which creates an encoding of the form

$$[v]_q = \frac{\alpha_1 \cdot \alpha_2 + \tilde{r}/(\beta_i \cdot \beta_j)}{z_i \cdot z_j} \mod q \tag{5}$$

where $\tilde{r} = \alpha_1 \cdot r_2 \cdot \beta_j + \alpha_2 \cdot r_1 \cdot \beta_i + (r_1 \cdot r_2)$.

As with GGH13, operations can only take place while the noise stays smaller than a set upper bound (in this case $\sqrt[\kappa]{q}$). We choose parameters such that κ multiplications can be computed without overflowing this boundary.

Zero-testing. To enable zero-testing on encodings $v_{\mathcal{U}}$ that are indexed at the top-level we follow procedures set in previous GE schemes and publish a zero-testing parameter p_{zt}. We consider top-level encodings that are constructed via a sequence of multiplications of encodings indexed with each of the sets $\mathcal{S}_i$ and thus they take the form:

$$[v_{\mathcal{U}}]_q = \frac{\tilde{\alpha} + \tilde{r}/\beta_{\mathcal{U}}}{z_{\mathcal{U}}}$$

where $\beta_{\mathcal{U}} = \prod_{i=1}^{\kappa} \beta_i$, $z_{\mathcal{U}} = \prod_{i=1}^{\kappa} z_i$ and $\tilde{\alpha}$ is a polynomial of underlying α values. Finally, $\tilde{r}$ is a polynomial taking the form $\hat{r} + \beta^{(1)} + \beta^{(2)} + \ldots + \beta^{(\kappa-1)}$ where we stratify the polynomial into the components $\beta^{(h)}$, containing all monomials of degree h in the β_i elements, and $\hat{r}$ representing a polynomial in the r_j values from each of the underlying encodings v_j. Notice that the polynomial structure of $\hat{r}$ exactly mirrors the polynomial that has been calculated over the encodings as a whole. The zero-test parameter is defined as

$$p_{zt} = \prod_{i=1}^{\kappa} \beta_i \cdot z_i$$

and an encoding $v_{\mathcal{U}}$ is zero-tested by first computing $\delta = p_{zt} \cdot v_{\mathcal{U}} \mod q$ and then we state that $v_{\mathcal{U}}$ encodes the value '0' if δ is 'small' and encodes a non-zero value if δ is 'big'.

Notice that

$$\delta_1 = \prod_{i=1}^{\kappa} \beta_i \cdot \tilde{\alpha} + \hat{r} + \beta^{(1)} + \beta^{(2)} + \ldots + \beta^{(\kappa-1)} \tag{6}$$

for non-zero encodings ($\tilde{\alpha} \neq 0$) and so for an encoding of zero ($\tilde{\alpha} \neq 0$) we have that

$$\delta_0 = \hat{r} + \beta^{(1)} + \beta^{(2)} + \ldots + \beta^{(\kappa-1)}. \tag{7}$$

The difference between Equations (6) and (7) is the loss of a factor of $\prod_{i=1}^{\kappa} \beta_i$, in Equation (7) we have monomials in the β_i of maximum degree $\kappa - 1$.

Observe that the value of the encoding is now stored in the MSB of the final output — in GGH13 the value is stored in the LSB. Therefore, zero-testing requires more involved distinguishing in our case than in GGH13. We discuss this in detail below.

Correctness. Correctness of zero-testing follows providing that

$$q^{\kappa-1/\kappa} < \|\delta_1\|_\infty < q, \quad \text{and} \quad q^{\kappa-2/\kappa} < \|\delta_0\|_\infty < q^{\kappa-1/\kappa}.$$

For δ_0, $\beta^{(1)} + \beta^{(2)} + \ldots + \beta^{(\kappa-1)}$ is a sum of monomials dominated by the term $\beta^{\kappa-1}$. In this particular term, we have a sum of monomials of degree $\kappa - 1$ over variables β_i, sampled such that $\|\beta_i\|_\infty < \sqrt[\kappa]{q}$. The coefficients of these monomials are made up of polynomials in the α, r elements from underlying encodings and are thus small. Therefore, providing that $\kappa \ll q$, we have that $q^{\kappa-2/\kappa} < \|\beta^{\kappa-1}\|_\infty < q^{\kappa-1/\kappa}$. Therefore, by appropriate choice of q, κ, we have that

$$q^{\kappa-2/\kappa} < \|\beta^{(1)} + \beta^{(2)} + \ldots + \beta^{(\kappa-1)}\|_\infty < q^{\kappa-1/\kappa}.$$

Again, since $\hat{r}_x$ is a monomial made up of variable sampled as small elements from R_q, then we have that δ_0 satisfies the relation above. Finally, since δ_1 contains a product of all β_i values then $q^{\kappa-1/\kappa} \leq \|\delta_1\|_\infty \leq q$.

We could set out tighter bounds for correctness but we are more concerned with the analysis of this scheme in the IO setting. As a consequence we merely state that it is possible to choose parameters such that distinguishing non-zero and zero encodings given how we sample each β_i.

5 Security analysis in IO setting

In this Section we demonstrate flaws in the encoding scheme from Section 4 via an annihilation attack in a simplified security model. Our attack does not depend on finding representations of any ideals. We will then highlight explicit machinery in IO constructions that may prevent the attack and may allow us to instantiate BGK-style obfuscators where annihilation attacks are not possible.

5.1 Simplified security model

In the attacks that we propose we prefer to talk explicitly in a game-based representation of the IO game. In particular, the scope of the weakened graded encoding model is unnecessarily wide since our attacks only occur during the post-zero-testing query phase. As a result we propose two interactive distinguishing security games. In the first, **IND-$\mathcal{M}$**, the adversary chooses functionally equivalent (Definition 6) branching programs to be encoded and then has oracle access for querying inputs on the branching program and receiving zero-tested outputs. In the second, **IND-OBF**, the branching programs are also randomised using the techniques (Kilian randomisation and multiplicative bundling scalars) discussed in Section 2.5 before encoding takes place — noting else is changed. In the weakened graded encoding model, the adversary is able to interact with random handles that represent encodings after operations have taken place.

Let

$$\mathsf{Adv}_{\mathcal{A}}^{\textbf{IND-}\mathcal{M}}(\lambda) = \left|\Pr\left[1 \leftarrow \mathcal{A}(\lambda, \widehat{\mathcal{M}}_0)\right] - \Pr\left[1 \leftarrow \mathcal{A}(\lambda, \widehat{\mathcal{M}}_1)\right]\right|$$

be the advantage of the adversary in **IND-$\mathcal{M}$**. Then the security game is satisfied if $\mathsf{Adv}_{\mathcal{A}}^{\textbf{IND-}\mathcal{M}}(\lambda) = \mathsf{negl}(\lambda)$. We give a formal description of the game in Figure 1.

Game $\mathbf{IND}\text{-}\mathcal{M}^{\mathcal{A}}(\lambda)$:	Oracle $\mathcal{O}_{\mathsf{zt}}(x)$:
1. $(\mathsf{sk}, \mathsf{prms}, \mathsf{evk}, \mathsf{ztk}) \leftarrow \mathsf{JInstGen}(1^\lambda, 1^\kappa)$	1. **if** init, $q \leftarrow 0$; **else**, $q \leftarrow q+1$
2. $(st, \mathcal{M}_0, \mathcal{M}_1) \leftarrow \mathcal{A}_0(l, \mathsf{prms}, \mathsf{evk})$	2. **if** $q > \mathcal{Q}$, $\delta \leftarrow \perp$
3. $b \leftarrow \{0,1\}$	3. **else**:
4. $\widehat{\mathcal{M}}_b \leftarrow \mathsf{JEnc}(\mathsf{sk}, \mathsf{prms}, \{\mathcal{S}_i\}_{i\in[\kappa]}, \mathcal{M}_b)$	4. $(\mathcal{U}, \mu_x) \leftarrow \widehat{\mathcal{M}}_b(x)$
5. $b' \leftarrow \mathcal{A}_1^{\mathcal{O}_{\mathsf{zt}}}(st)$	5. $\delta_x \leftarrow \mathsf{JZTParam}(\mathsf{prms}, \mathsf{ztk}, (\mathcal{U}, \mu_x))$
6. output $(b' = b)$	6. return δ_x

Fig. 1. Left: The $\mathbf{IND}\text{-}\mathcal{M}$ game. An adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ is legitimate if $\mathcal{A}_0$ outputs two branching programs $(\mathcal{M}_0, \mathcal{M}_1)$ of the same size that compute functionally equivalent circuits. We abuse notation and write $\mathsf{JEnc}(\mathsf{sk}, \mathsf{prms}, \{\mathcal{S}_i\}_{i\in[\kappa]}, \mathcal{M}_b)$ to denote the encoding of the i^{th} level of matrices in $\mathcal{M}_b$ with respect to the index set $\mathcal{S}_i$. **Right**: Oracle for computing inputs on the encoded branching program $\widehat{\mathcal{M}}_b$ and outputting zero-tested results.

Let

$$\mathsf{Adv}^{\mathbf{IND\text{-}OBF}}_{\mathcal{A}}(\lambda) = \left|\Pr\left[1 \leftarrow \mathcal{A}(\lambda, \widehat{\mathcal{M}_0})\right] - \Pr\left[1 \leftarrow \mathcal{A}(\lambda, \widehat{\mathcal{M}_1})\right]\right|$$

be the advantage of the adversary in **IND-OBF**, the security game is satisfied if $\mathsf{Adv}^{\mathbf{IND\text{-}OBF}}_{\mathcal{A}}(\lambda) = \mathsf{negl}(\lambda)$. We give a formal description of the game in Figure 2; we use iO to denote the oracle that the challenger uses for performing the BGK-style obfuscation of a branching program.

Game $\mathbf{IND}\text{-}\mathcal{M}^{\mathcal{A}}(\lambda)$:	Oracle $\mathcal{O}_{\mathsf{zt}}(x)$:
1. $(\mathsf{sk}, \mathsf{prms}, \mathsf{evk}, \mathsf{ztk}) \leftarrow \mathsf{JInstGen}(1^\lambda, 1^\kappa)$	1. **if** init, $q \leftarrow 0$; **else**, $q \leftarrow q+1$
2. $(st, \mathcal{M}_0, \mathcal{M}_1) \leftarrow \mathcal{A}_0(l, \mathsf{prms}, \mathsf{evk})$	2. **if** $q > \mathcal{Q}$, $\delta \leftarrow \perp$
3. $b \leftarrow \{0,1\}$	3. **else**:
4. $\widehat{\mathcal{M}}_b \leftarrow \mathsf{iO}(\mathsf{sk}, \mathsf{prms}, \{\mathcal{S}_i\}_{i\in[\kappa]}, \mathcal{M}_b)$	4. $(\mathcal{U}, \mu_x) \leftarrow \widehat{\mathcal{M}}_b(x)$
5. $b' \leftarrow \mathcal{A}_1^{\mathcal{O}_{\mathsf{zt}}}(st)$	5. $\delta_x \leftarrow \mathsf{JZTParam}(\mathsf{prms}, \mathsf{ztk}, (\mathcal{U}, \mu_x))$
6. output $(b' = b)$	6. return δ_x

Fig. 2. Left: The **IND-OBF** game. Same as above, except that the oracle iO takes a branching program as input and outputs an obfuscation of the branching program, as defined in Section 2.5. The i^{th} level of matrices is still encoded with respect to $\mathcal{S}_i$. **Right**: Oracle for computing inputs on the encoded branching program $\widehat{\mathcal{M}}_b$ and outputting zero-tested results.

5.2 Analysis of (in)security in IND-$\mathcal{M}$ game

Let $\widehat{\mathcal{M}_b}$ be the encoded branching program that $\mathcal{A}_1$ receives in the $\mathbf{IND}\text{-}\mathcal{M}$ security game. Then let $\mathcal{X} = \{0,1\}^L$ be the set of valid inputs to $\widehat{\mathcal{M}}_b$ and let $\kappa = L+2$ be the total degree of multilinearity (due to encoding of bookends as well). Let $\mu_x = \widehat{\mathcal{M}}_b(x)$ be the output of $\widehat{\mathcal{M}}_b$ on some input $x \in \{0,1\}^L$ and let $\delta_x = \mathcal{O}_{\mathsf{zt}}(\mu_x)$. Assuming that μ_x is honestly computed then δ_x should be meaningful.

Here we show that if μ_x is generated honestly and that if $\mathsf{JTest}(\delta_x) = 1$ then it is possible to distinguish which branching program has been encoded. Our analysis uses an annihilation attack that is very similar in spirit to the original given in [MSZ16a]. Let $\widehat{M}_{x_{\mathsf{inp}(l)},l}$ be a matrix at level l in $\widehat{\mathcal{M}}_b$. Let $\alpha^{x_{\mathsf{inp}(l)}}_{i,j,l} = M_{x_{\mathsf{inp}(l)},l}[i][j]$ in the original branching program $\mathcal{M}_b$. Recall that the corresponding entry $\widehat{M}_{x_{\mathsf{inp}(l)},l}[i][j]$ after encoding has taken place takes the form:

$$v^{x_{\mathsf{inp}(l)}}_{i,j,l} = (\alpha^{x_{\mathsf{inp}(l)}}_{i,j,l} + r^{x_{\mathsf{inp}(l)}}_{i,j,l}/\beta_l)/z_l.$$

In the attack we treat the variables $r^{x_{\mathsf{inp}(l)}}_{i,j,l}$ as formal variables and show that, for enough inputs x, we can compute a polynomial Q that annihilates these variables. Let r^x denote the set of all variables $r^{x_{\mathsf{inp}(l)}}_{i,j,l}$ that are used in computing $\widehat{\mathcal{M}}_b(x)$.

Firstly notice that the form of δ_x is the following:

$$\delta_x = \hat{r}_x + \beta^{(1)} + \beta^{(2)} + \ldots + \beta^{(\kappa-1)}$$

recalling that $\hat{r}_x$ is a polynomial dependent only on r_j values from encodings and $\beta^{(\ell)}$ is the sum of all monomials in the terms $\{\beta_i\}_{i\in[\kappa]}$ of degree ℓ. In this analysis we focus on the term $\beta^{(\kappa-1)}$, which has coefficients in the $\alpha^{x_{\mathsf{inp}(l)}}_{i,j,l}, r^{x_{\mathsf{inp}(l)}}_{i,j,l}$ terms. The adversary $\mathcal{A}$ knows the values $\alpha^{x_{\mathsf{inp}(l)}}_{i,j,l}$ as they are the entries from one of the original branching programs. The values β_i are the same in both branching programs so we can view $\beta^{(\kappa-1)}$ as a linear polynomial in the set of variables $r^{(x)}$.

Now, consider two bit strings x, x' where there is only a single bit difference between the two at position $t \neq 1$. Let $x_l = x_{\mathsf{inp}(l)} = x'_{\mathsf{inp}(l)}$ and consider the monomials in the matrix entries $r^{x_{\mathsf{inp}(1)}}_{i,j,1}$. Notice that the coefficients will only change in the values $\alpha^{x_{\mathsf{inp}(t)}}_{i,j,t}$ — since the β_l variables are fixed at level l. Let $c^{x_{\mathsf{inp}(1)}}_{i,j,1}$, $c'^{x_{\mathsf{inp}(1)}}_{i,j,1}$ denote the coefficients of $r^{x_{\mathsf{inp}(1)}}_{i,j,1}$ after computing $\delta_x \leftarrow \mathsf{JZTParam}(\widehat{\mathcal{M}}_b(x))$ and $\delta_{x'} \leftarrow \mathsf{JZTParam}(\widehat{\mathcal{M}}_b(x'))$, respectively. Then, as long as $\mathsf{JTest}(\delta_x) = \mathsf{JTest}(\delta_{x'}) = 1$, notice that if we compute

$$c'^{x_{\mathsf{inp}(1)}}_{i,j,1}\delta_x - c^{x_{\mathsf{inp}(1)}}_{i,j,1}\delta_{x'} = \tilde{\delta}.$$

we remove the monomial in the variable $r^{x_{\mathsf{inp}(1)}}_{i,j,1}$. To see this, note that by multiplying through the coefficient of $r^{x_{\mathsf{inp}(1)}}_{i,j,1}$ with the coefficient from the opposing output gives equal monomials in both expressions. Subtracting the two scaled outputs removes this monomial entirely, all other monomials are scaled by the coefficient that is multiplied through.

The power of this attack is that it only requires four inputs to remove all monomials that are linear in some $r^{x_{\mathsf{inp}(l)}}_{i,j,l}$ for any program length L. That is, we iterate over the possible four choices for the first two input bits and fix the remaining $L-2$ input bits. Recall, that we are only interested in annihilating the monomials that are linear in the variables $r^{x_{\mathsf{inp}(l)}}_{i,j,l}$. Moreover, the annihilation is performed on the polynomial evaluations that arise after computing the branching program on these four different inputs. Let $x \in \{00, 01, 10, 11\}$ describe the four varying inputs and notice that, for any $r^{x_{\mathsf{inp}(l')}}_{i,j,l'}$ for $l' \notin \{1, 2\}$, then the coefficient of the monomial in δ_x changes only in the values of $\alpha^{x_{\mathsf{inp}(l_x)}}_{i,j,l_x}$ for $l_x \in \{1, 2\}$ — which are known the adversary. Moreover, for the monomials in entries $r^{x_{\mathsf{inp}(l_x)}}_{i,j,l_x}$, precisely two of the polynomial evaluations contain monomials with differing coefficients. Therefore, we have at least two polynomials containing each variable $r^{x_{\mathsf{inp}(l)}}_{i,j,l}$; thus the problem essentially becomes solving a set of linear simultaneous equations of rank less than four. By performing operations over these polynomial evaluations to reduce the rank of the system from four to three then we create a polynomial where all monomials $r^{x_{\mathsf{inp}(l)}}_{i,j,l}$ are annihilated. There are $25(L+2)+10$ variables that need to be annihilated which is linear in the length of the branching program — the attack needs to be applied this many times but this is clearly efficient.[8]

Thus, once an equation is solved a distinguishing attack in the **IND-$\mathcal{M}$** game can be launched. The final result is a term δ that is noticeably smaller than all previous outputs δ_x since the only monomials that are left have degree $\leq \kappa - 2$ in the variables β_i. The β_i variables are the largest components of δ_x and so the magnitude noticeably decreases. Notice, that the attack only works for the choice of one branching program, i.e. the attack works in the case where $\widehat{\mathcal{M}}_0$ is encoded rather than $\widehat{\mathcal{M}}_1$ (without loss of generality). Therefore, if the attack fails (i.e. the magnitude of δ is no smaller) then $\mathcal{A}$ outputs $b' = 1$; if it does work they output $b' = 0$. The attack works with probability 1 and so we have no security when analysing our MJP scheme in the **IND-$\mathcal{M}$** game.

5.3 Initial analysis of security in IND-OBF game

Recall that the **IND-OBF** game adds extra randomisation details to the branching program that are commonplace in most IO constructions. Importantly, it mimics the structure of BGK-style obfuscated branching programs. In this setting, instantiating using GGH13 is proven insecure based on the attacks

[8] After each iteration the output is also scaled by the coefficients used previously so these need to be taken account for in further operations.

of [CGH17, MSZ16a, ADGM16] and as such these attacks also work in our simplified model. We show that there is an observable difference between our MJP scheme and that of GGH13 since the multiplicative bundling scalars introduced in **IND-OBF** seem to render our previous attack useless. However, we stress that it is likely a variant of the attack could be used to still distinguish easily in the game.

Concretely, the adversary in **IND-OBF** receives the matrices $\widehat{M}_0, \widehat{M}_{x_{\mathsf{inp}(l)},l}, \widehat{M}_{L+1}$ that are, respectively, encodings of the following matrices:

- $\epsilon_0 M_0 \cdot \mathcal{R}_0$;
- $\epsilon_{x_{\mathsf{inp}(l)},l}\mathcal{R}_{l-1}^{-1} \cdot M_{x_{\mathsf{inp}(l)},l} \cdot \mathcal{R}_l$;
- $\epsilon_{L+1}\mathcal{R}_L^{-1} \cdot M_{L+1}$.

Recall, $\mathcal{R}_0, \ldots, \mathcal{R}_L$ are randomly sampled invertible matrices used to implement Kilian's randomisation technique and $\epsilon_0, \{\epsilon_{x_{\mathsf{inp}(l)},l}\}_{l\in[L]}, \epsilon_{L+1}$ are random multiplicative bundling scalars taken from R_q. There are other techniques that are used for protecting constructions of indistinguishability obfuscation (such as encoding with respect to a straddling sets structure), but for simplicity we only consider these randomisation measures.

Indeed it appears that the randomisation matrices render the encoded values obscure to the adversary. However we can write each encoded matrix at level l in the form:

$$\widehat{M}_{x_{\mathsf{inp}(l)},l} = \epsilon_{x_{\mathsf{inp}(l)},l}\mathcal{R}_{l-1}^{-1} \cdot M_{x_{\mathsf{inp}(l)},l} \cdot \mathcal{R}_l + E_{x_{\mathsf{inp}(l)},l}/\beta_l$$

where $E_{x_{\mathsf{inp}(l)},l}$ is a matrix containing the entries $r_{i,j,l}^{x_{\mathsf{inp}(l)}}$.[9] However, a change of variables transformation allows us to rewrite the encodings as

$$\widehat{M}_{x_{\mathsf{inp}(l)},l} = \epsilon_{x_{\mathsf{inp}(l)},l}\mathcal{R}_{l-1}^{-1} \cdot (M_{x_{\mathsf{inp}(l)},l} + E_{x_{\mathsf{inp}(l)},l}/\beta_l) \cdot \mathcal{R}_l$$

which lets us assume that the adversary still has knowledge of the encoded values. This technique was first used by Miles et al. [MSZ16a] in justifying their annihilation attack scenario. Thus, while Kilian randomisation procedures are still regarded as an important facet of IO candidates, it would appear that they do very little to prevent known attacks.[10]

The reason that our attack no longer works trivially is due to the introduction of the multiplicative bundling scalars. These act as unknown variables in the final output when computing δ_x as the zero-tested output of $\widehat{\mathcal{M}}_b(x)$. Consider the monomials that are linear in some random variable $r_{i,j,l}^{x_{\mathsf{inp}(l)}}$; we previously annihilated any such monomial by picking two inputs x, x' differing only on one input bit in position t. The outputs δ_x, δ_x' would now also include different scalars $\epsilon_{x_t,t}$ that are unknown to the adversary. This means that the previous technique for annihilating the monomial in variable $r_{i,j,l}^{x_{\mathsf{inp}(l)}}$ is no longer possible since it would also require scaling by $\epsilon_{x_t,t}$.

There are $2L+2$ input mixing scalars and a total $50L+10$ matrix entries $r_{i,j,l}^{x_{\mathsf{inp}(l)}}$. It is, in principle, possible (by a corollary of Theorem 1 [Kay09]) to carry out a generic annihilation attack since there are 2^L possible inputs. Such an attack would require finding annihilating polynomials for each of the monomials (degree $\kappa+1$) in the variables $r_{i,j,l}^{x_{\mathsf{inp}(l)}}, \{\epsilon_{x_{\mathsf{inp}(l)},l}\}_l, \epsilon_0, \epsilon_{L+1}$ for an input x. Computing general annihilating polynomials is believed to be a $\#P$-hard problem and indeed there are some sets of cubic polynomials for which the annihilating polynomial cannot be expressed by a poly-depth circuit [MSZ16a, Kay09]. As such, the distinguishing attack that we use above no longer seems to apply, though we stress that we cannot prove hardness under any reasonable assumption.

[9] We ignore the usage of each z_l in the encodings for now as these are removed after zero-testing.

[10] It may be wise to no longer think of these random invertible matrices as offering any security when analysing IO candidates.

Thus, it appears that our encoding scheme may still be able to be used in instantiating BGK-style obfuscators in our security model, even though these obfuscators are insecure when instantiated via GGH13. Exploring the possibility of attacks that could distinguish in this setting would be valuable in further ascertaining the relationship between our encodings and original GGH13.

6 Discussion of findings

Finally, we summarise and address the key points arising from our analysis and give points that may warrant further attention.

Structural faults in GGH13. Our main focus is to highlight that the GGH13 GES bears structural faults that are vulnerable even when a natural variant where the ideals are removed is constructed. All previous attacks exploit the presence of the generator g in each encoding to learn a basis of the ideal $\langle g \rangle$. We show that removing the capability to learn this ideal does not prevent attacks that are able to distinguish between encoded branching programs in a simplified model.

These faults appear to be mitigated by considering more complete obfuscation techniques in our security model, though we fully expect a more sophisticated attack to be viable via the annihilating polynomial route. In order to make security more concrete we would need to reduce the hardness of finding annihilating polynomials for our case back to a plausible hard problem. This may take into account the hardness of finding an annihilating polynomial for a general set of algebraically dependent polynomials in R_q.

CGH attacks. The attacks of Chen, Gentry and Halevi [CGH17] use a variant of an annihilation attack, along with knowledge of the ratios of input mixing scalars to launch powerful attacks on various IO candidates (including recent immunisations). These attacks can be prevented using input authentication methods [FRS16], however these prevention methods lie outside scope of the weakened graded encoding model.

It is not completely clear whether a variant of the CGH attack can be leveraged on an IO candidate using our MJP scheme. This is because it explicitly launches a distinguishing attack based on the ideal $\langle g \rangle$. It would be valuable to investigate whether a variant of their attack can be used to break our MJP scheme as well. Such a result would add weight to the fact that structural faults in 'GGH-like' encodings are to be blamed rather than the presence of g explicitly.

Thus, it might be preferable to instantiate immunised IO candidates with our MJP scheme rather than GGH13 as our encodings provide no natural way of finding a basis of representative ideals. Furthermore, immunised constructions seem to prevent MSZ-like attacks on the underlying encoding schemes and thus also prevent the simplified attack that we described in Section 5.2. However, this should not be construed as having confidence in the security of our scheme. It is an interesting open question if the attacks in [CGH17] can be generalised to our case.

References

ABD16. 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 Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 153–178. Springer, Heidelberg, August 2016.

ADGM16. Daniel Apon, Nico Döttling, Sanjam Garg, and Pratyay Mukherjee. Cryptanalysis of indistinguishability obfuscations of circuits over ggh13. Cryptology ePrint Archive, Report 2016/1003, 2016. http://eprint.iacr.org/2016/1003.

AGIS14. Prabhanjan Vijendra Ananth, Divya Gupta, Yuval Ishai, and Amit Sahai. Optimizing obfuscation: Avoiding Barrington's theorem. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 14*, pages 646–658. ACM Press, November 2014.

Bar89. David A. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in nc^1. *J. Comput. Syst. Sci.*, 38(1):150–164, 1989.

BGK+14. 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 2014*, volume 8441 of *LNCS*, pages 221–238. Springer, Heidelberg, May 2014.

BLR+15. Dan Boneh, Kevin Lewi, Mariana Raykova, Amit Sahai, Mark Zhandry, and Joe Zimmerman. Semantically secure order-revealing encryption: Multi-input functional encryption without obfuscation. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 563–594. Springer, Heidelberg, April 2015.

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

BS83. Walter Baur and Volker Strassen. The complexity of partial derivatives. *Theor. Comput. Sci.*, 22:317–330, 1983.

BWZ14. Dan Boneh, Brent Waters, and Mark Zhandry. Low overhead broadcast encryption from multilinear maps. In Garay and Gennaro [GG14], pages 206–223.

CGH17. Yilei Chen, Craig Gentry, and Shai Halevi. Cryptanalyses of candidate branching program obfuscators. In *Advances in Cryptology - EUROCRYPT 2017*, pages 278–307, 2017.

CIV16. Wouter Castryck, Ilia Iliashenko, and Frederik Vercauteren. Provably weak instances of ring-LWE revisited. In Fischlin and Coron [FC16], pages 147–167.

CJL16. Jung Hee Cheon, Jinhyuck Jeong, and Changmin Lee. An algorithm for ntru problems and cryptanalysis of the ggh multilinear map without a low-level encoding of zero. *LMS Journal of Computation and Mathematics*, 19(A):255–266, 2016.

CLLT16. Jean-Sébastien Coron, Moon Sung Lee, Tancrède Lepoint, and Mehdi Tibouchi. Cryptanalysis of GGH15 multilinear maps. In Robshaw and Katz [RK16], pages 607–628.

CLLT17. Jean-Sébastien Coron, Moon Sung Lee, Tancrède Lepoint, and Mehdi Tibouchi. Zeroizing attacks on indistinguishability obfuscation over CLT13. In Serge Fehr, editor, *PKC 2017, Part I*, volume 10174 of *LNCS*, pages 41–58. Springer, Heidelberg, March 2017.

CLR15. Jung Hee Cheon, Changmin Lee, and Hansol Ryu. Cryptanalysis of the new CLT multilinear maps. Cryptology ePrint Archive, Report 2015/934, 2015. `http://eprint.iacr.org/2015/934`.

CLT13. Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Practical multilinear maps over the integers. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 476–493. Springer, Heidelberg, August 2013.

CLT14. Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Cryptanalysis of two candidate fixes of multilinear maps over the integers. Cryptology ePrint Archive, Report 2014/975, 2014. `http://eprint.iacr.org/2014/975`.

FC16. Marc Fischlin and Jean-Sébastien Coron, editors. *EUROCRYPT 2016, Part I*, volume 9665 of *LNCS*. Springer, Heidelberg, May 2016.

FRS16. Rex Fernando, Peter M. R. Rasmussen, and Amit Sahai. Preventing clt zeroizing attacks on obfuscation. Cryptology ePrint Archive, Report 2016/1070, 2016. `http://eprint.iacr.org/2016/1070`.

GG14. Juan A. Garay and Rosario Gennaro, editors. *CRYPTO 2014, Part I*, volume 8616 of *LNCS*. Springer, Heidelberg, August 2014.

GGH13a. Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 1–17. Springer, Heidelberg, May 2013.

GGH+13b. Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th FOCS*, pages 40–49. IEEE Computer Society Press, October 2013.

GGH+13c. Sanjam Garg, Craig Gentry, Shai Halevi, Amit Sahai, and Brent Waters. Attribute-based encryption for circuits from multilinear maps. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 479–499. Springer, Heidelberg, August 2013.

GGH15. Craig Gentry, Sergey Gorbunov, and Shai Halevi. Graph-induced multilinear maps from lattices. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 498–527. Springer, Heidelberg, March 2015.

GMM+16. Sanjam Garg, Eric Miles, Pratyay Mukherjee, Amit Sahai, Akshayaram Srinivasan, and Mark Zhandry. Secure obfuscation in a weak multilinear map model. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 241–268. Springer, Heidelberg, October / November 2016.

GMS16. Sanjam Garg, Pratyay Mukherjee, and Akshayaram Srinivasan. Obfuscation without the vulnerabilities of multilinear maps. Cryptology ePrint Archive, Report 2016/390, 2016. `http://eprint.iacr.org/2016/390`.

Hal15. Shai Halevi. Graded encoding, variations on a scheme. Cryptology ePrint Archive, Report 2015/866, 2015. `http://eprint.iacr.org/2015/866`.

HJ16. Yupu Hu and Huiwen Jia. Cryptanalysis of GGH map. In Fischlin and Coron [FC16], pages 537–565.

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

KF17. Paul Kirchner and Pierre-Alain Fouque. Revisiting lattice attacks on overstretched NTRU parameters. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology - EUROCRYPT 2017, Proceedings, Part I*, volume 10210 of *Lecture Notes in Computer Science*, pages 3–26, 2017.

Kil88. Joe Kilian. Zero-knowledge with log-space verifiers. In *29th FOCS*, pages 25–35. IEEE Computer Society Press, October 1988.

LTV12. Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In Howard J. Karloff and Toniann Pitassi, editors, *44th ACM STOC*, pages 1219–1234. ACM Press, May 2012.

MR04. Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on Gaussian measures. In *45th FOCS*, pages 372–381. IEEE Computer Society Press, October 2004.

MSW14. Eric Miles, Amit Sahai, and Mor Weiss. Protecting obfuscation against arithmetic attacks. Cryptology ePrint Archive, Report 2014/878, 2014. `http://eprint.iacr.org/2014/878`.

MSZ16a. Eric Miles, Amit Sahai, and Mark Zhandry. Annihilation attacks for multilinear maps: Cryptanalysis of indistinguishability obfuscation over GGH13. In Robshaw and Katz [RK16], pages 629–658.

MSZ16b. Eric Miles, Amit Sahai, and Mark Zhandry. Secure obfuscation in a weak multilinear map model: A simple construction secure against all known attacks. Cryptology ePrint Archive, Report 2016/588, 2016. `http://eprint.iacr.org/2016/588`.

PST14. Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation from semantically-secure multilinear encodings. In Garay and Gennaro [GG14], pages 500–517.

RK16. Matthew Robshaw and Jonathan Katz, editors. *CRYPTO 2016, Part II*, volume 9815 of *LNCS*. Springer, Heidelberg, August 2016.

A MJP from our encoding scheme

Using the encodings that we describe in Section 4.1 we can now construct an MJP scheme. Note that we refer to jigsaw generation and verification as both algorithms and as specific roles within a computation interchangeably.

A.1 Setup

Instance generation. ($\mathsf{JInstGen}$): On input the security parameter 1^λ, and perceived multilinearity κ the algorithm does the following:

- Samples the prime integer q
- Samples κ uniform polynomials z_i from the ring R_q
- Samples κ polynomials β_i fulfilling the requirements set out in Section 4
- Outputs $(\mathsf{sk}, \mathsf{pp}) = ((\beta_i, z_i), (\{\mathcal{S}_i\}_{i\in[\kappa]}, q))$

Encoding. (JEnc): This algorithm takes as input some value α, an index set $\mathcal{S}_i$ and a pair $(\beta_i, z_i) = \mathsf{sk}$ sampled from $\mathsf{JInstGen}$ and:

- Samples a small element r uniformly from the error distribution χ
- Computes $v = \frac{\alpha + r/\beta_i}{z_i}$ as an encoding of the value α

Jigsaw generation. (JGen): Takes as input an index set $\mathcal{S}_i$, the pair (β_i, z_i) for $i \in [\kappa]$ and associated encoded values $(\alpha_1, \ldots, \alpha_{m_i})$ for each of these pairs, where m_i is the number of values to be encoded with respect to $\mathcal{S}_i$. Then this algorithm performs the following:

- Inputs each tuple $(\mathcal{S}_i, \alpha_j)$ for $j \in [m_i]$ to the encode algorithm JEnc and receives back the κ sets C_i where C_i consists of all pairs $(\mathcal{S}_i, v_j)$ for $1 \le j \le m_i$.

– Generates the zero-testing parameter p_{zt} by computing

$$p_{zt} = \prod_{i=1}^{\kappa} \beta_i \cdot z_i$$

.

– Creates

$$\mathsf{puzzle} = (q, \{C_1, \ldots, C_\kappa\}, p_{zt}) \tag{8}$$

as the public output. Let $\boldsymbol{\alpha}^{(i)}$ be the set of values $\{\alpha_1, \ldots, \alpha_{m_i}\}$ that are encoded with respect to $\mathcal{S}_i$ — then the private output is defined as

$$X = (\boldsymbol{\alpha}^{(1)}, \ldots, \boldsymbol{\alpha}^{(\kappa)}).$$

Note that the values r, β and z are all kept secret in order to preserve the secrecy of the encoded values. Public access to each β_i and z_i is granted in the form of the zero-test parameter p_{zt}, though it should be impossible to decompose this into the individual factors.

A.2 Jigsaw verification

As before, we note that the zero-test procedure is split into three separate algorithms to accurately model the security setting that we consider. These three algorithms are defined as the following:

Computation. (JCompute): Takes as input the encodings $v_j^{(i)}$ with respect to each index set $\mathcal{S}_i$ and a multilinear form, F, and outputs

$$v^* = F(v_1^{(1)}, \ldots, v_{m_1}^{(1)}, \ldots, v_1^{(\kappa)}, \ldots, v_{m_\kappa}^{(\kappa)})$$

where v^* is a top-level encoding as shown in Equation (5).

Zero-testing. (JZTParam): Takes as input an encoding v^* resulting from the JCompute algorithm and p_{zt} from puzzle and output $\delta = p_{zt} \cdot v^*$

Zero-test output. (JTest): Takes δ as an output from the JZTParam algorithm and checks the magnitude of the element. If it has magnitude greater than $\prod_{i=1}^{\kappa} \beta_i$ then output 1 (encoded value is zero). Otherwise output 0 (encoded value is non-zero).

Finally the overarching JVer algorithm defined previously simply runs these three algorithms in sequence and outputs the result of JTest.

A.3 Correctness of construction

The homomorphic properties of our encodings as shown in the previous section enable us to evaluate the multilinear forms that are input to the JCompute algorithm. Correctness is lost post-zero-testing if wrap-around modulo q occurs for a top-level encoding, or if an encoding of zero exceeds $q^{\frac{\kappa-1}{\kappa}}$. Since we specifically sample the β_i elements from R_q such that we satisfy these requirements and since each of the sampled elements are small.