# Zeroizing Attacks on Indistinguishability Obfuscation over CLT13

Jean-Sébastien Coron[1], Moon Sung Lee[1],
Tancrède Lepoint[2], and Mehdi Tibouchi[3]

[1] University of Luxembourg
[2] SRI International
[3] NTT Secure Platform Laboratories

**Abstract.** In this work, we describe a new polynomial-time attack on the multilinear maps of Coron, Lepoint, and Tibouchi (CLT13), when used in candidate iO schemes. More specifically, we show that given the obfuscation of the simple branching program that computes the always zero functionality previously considered by Miles, Sahai and Zhandry (Crypto 2016), one can recover the secret parameters of CLT13 in polynomial time via an extension of the zeroizing attack of Coron et al. (Crypto 2015). Our attack is generalizable to arbitrary *oblivious* branching programs for arbitrary functionality, and allows (1) to recover the secret parameters of CLT13, and then (2) to recover the randomized branching program entirely. Our analysis thus shows that several of the single-input variants of iO over CLT13 are insecure.

**Keywords:** Multilinear Maps, CLT13, Indistinguishability Obfuscation, Zeroizing Attacks.

## 1 Introduction

Since their introduction, all candidates for multilinear maps [GGH13a, CLT13, GGH15] have been shown to suffer from zeroizing attacks [GGH13a, CHL+15, GGH15], sometimes even when no low-level encoding of zero was made available to the adversary [CGH+15]. However, the leading application of multilinear maps, indistinguishability obfuscation, has until now remained little affected by this kind of attacks. This resistance seemed to come from the fact that the particular combinations enforced in indistinguishability obfuscation constructions did not allow enough freedom to obtain a simple system of successful zero-tests that could be solved using linear algebraic techniques; see the discussion on the limitations of zeroizing attacks in [CGH+15, Sec. 1.2].

**Attacks against iO (Related Work).** Attacks against simplified variants of certain obfuscation schemes instantiated over the Coron-Lepoint-Tibouchi (CLT13) multilinear maps [CLT13] have been described in [CGH+15]. Firstly, the GGHRSW branching-program (BP) obfuscation procedure from [GGH+13b] has been shown to be broken for branching programs with a special "decomposable" structure where the inputs bits can be partitioned in three sets, and so that one set only affects the first steps of the BP, a second set the middle steps of the BP, and the last set the final steps of the BP. Secondly, the simple variants of the circuit obfuscation procedures from [Zim15, AB15] has been shown to be broken for simple circuits, such as point functions.

Recently in [MSZ16], Miles, Sahai and Zhandry introduced annihilation attacks against multilinear maps, and applied them to cryptanalyze in polynomial-time several candidate iO schemes [BGK+14, MSW14, AGIS14, PST14, BMSZ16] over the Garg-Gentry-Halevi (GGH13) multilinear maps. The core idea of the attack against to differentiate whether an obfuscated program $\mathcal{O}$ comes from a branching program $\mathbf{A}$ or a branching program $\mathbf{A}'$ is the following: evaluate specific inputs $x_i$'s that evaluate to 0 on $\mathbf{A}$ and $\mathbf{A}'$, get the zero-tested values $y_i = O(x_i)$, and then evaluate an annihilating

polynomial $Q_{\mathbf{A}}$ constructed from $\mathbf{A}$ over the $y_i$'s. When $\mathbf{A}$ was obfuscated, $Q_{\mathbf{A}}(y)$ belongs to an ideal $\mathcal{I}$ independent of $y$ and $\mathbf{A}$; otherwise $Q_{\mathbf{A}}(y) \notin \mathcal{I}$ with high probability. Annihilation polynomials can also be used to attack the order revealing encryption scheme proposed in [BLR+15], but fall short of attacking the initial GGHRSW candidate $\mathsf{iO}$ scheme [GGH+13b].

**Our contributions.** In the remaining of the document, we cryptanalyze several constructions of indisguishability obfuscation [GGH+13b, MSW14, AGIS14, PST14, BMSZ16, GMM+16] when instantiated over CLT13. More specifically, we show the following theorem.

**Theorem 1.** *Let $\mathcal{O}$ denote the single-input variant of the $\mathsf{iO}$ candidates in [GGH+13b, MSW14, AGIS14, PST14, BMSZ16, GMM+16] (over CLT13 multilinear maps). There exists a branching program $\mathbf{A}$ such that, given $\mathcal{O}(\mathbf{A})$, one can break the CLT13 multilinear maps in polynomial-time.*

To show this, we use the branching program $\mathbf{A}$ that computes the always-zero function previously considered in [MSZ16], in which every matrix is simply the identity matrix. This branching program does not fit in the framework of the zeroizing attacks proposed in [CGH+15], but we show that one can reconstruct the three-ways structure required by the zeroizing attacks by using tensor products. More precisely, consider a branching program evaluation on input $x$

$$A(x) = \widehat{\boldsymbol{A}}_0 \times \prod_{i=1}^{2t} \widehat{\boldsymbol{A}}_{i,x_{\mathsf{inp}(i)}} \times \widehat{\boldsymbol{A}}_{2t+1} \times p_{zt} \bmod x_0\,,$$

where $\mathsf{inp}(i) = \min(i, 2t+1-i)$ denotes the input bit used at the $i$-th step of the computation and $\widehat{\boldsymbol{A}} = \{\widehat{\boldsymbol{A}}_0, \widehat{\boldsymbol{A}}_{2t+1}, \widehat{\boldsymbol{A}}_{i,b} \mid i \in [2t], b \in \{0,1\}\}$ is the obfuscated branching program. We show that $A(x)$ can be rewritten as a product of consecutive factors

$$\begin{aligned} A(x) &= \boldsymbol{B}(x) \times \boldsymbol{C}(x) \times \boldsymbol{D}(x) \times \boldsymbol{C}'(x) \times \boldsymbol{B}'(x) \times p_{zt} \bmod x_0 \\ &= \left(\boldsymbol{B}'(x)^T \otimes \boldsymbol{B}(x)\right) \times \left(\boldsymbol{C}'(x)^T \otimes \boldsymbol{C}(x)\right) \times \mathsf{vec}\left(\boldsymbol{D}(x)\right) \times p_{zt} \bmod x_0, \end{aligned}$$

where the factors $\boldsymbol{B}'(x)^T \otimes \boldsymbol{B}(x), \boldsymbol{C}'(x)^T \otimes \boldsymbol{C}(x)$ and $\boldsymbol{D}(x)$ that can be made to vary independently, and $\mathsf{vec}(\boldsymbol{D})$ denotes the vector formed by stacking the columns of the matrix $\boldsymbol{D}$ on top of each other. We then show how to extend the zeroizing attack approach described in [CHL+15, CGH+15] to construct a block diagonal matrix, and apply the Cayley-Hamilton theorem to recover all the secrets embedded in the CLT13 public parameters. Once the multilinear map secret parameters have been recovered, one can then *recover the randomized branching program* $\widehat{\mathbf{A}}$ completely. Thus, one can distinguish between the obfuscation of two branching programs whenever they are inequivalent under Kilian's randomization.

Our attack is applicable to the single-input version of the candidate obfuscators from [MSW14, AGIS14, PST14, BMSZ16], to the GGHRSW obfuscator [GGH+13b] (as opposed to annihilations attacks), but also to the obfuscator [GMM+16] proved secure in the weak multilinear map model (therefore preventing annihilation attacks).

Last, but not least, we then show how to *generalize* our attack to branching programs with an essentially arbitrary structure, including oblivious branching programs, and to programs achieving essentially arbitrary functionalities. This shows that the previously mentioned single-input obfuscators should be considered broken when instantiated with CLT13.

## 2 Preliminaries

*Notation.* We use $[a]_n$ or $a \bmod n$ to denote a unique integer $x \in (-\frac{n}{2}, \frac{n}{2}]$ which is congruent to $a$ modulo $n$. A set $\{1, 2, \ldots, n\}$ is denoted by $[n]$. Vectors and matrices will be denoted by bold letters. The transpose of a matrix $\boldsymbol{A}$ is denoted by $\boldsymbol{A}^T$.

### 2.1 Kronecker product of matrices

For any two matrices $\boldsymbol{A} \in R^{m \times n}$ and $\boldsymbol{B} \in R^{p \times q}$, we define the *Kronecker product* (or tensor product) of $\boldsymbol{A}$ and $\boldsymbol{B}$ as the block matrix $\boldsymbol{A} \otimes \boldsymbol{B} \in R^{(mp) \times (nq)}$ given by:

$$\boldsymbol{A} \otimes \boldsymbol{B} = \begin{bmatrix} a_{11}\boldsymbol{B} & \cdots & a_{1n}\boldsymbol{B} \\ \vdots & \ddots & \vdots \\ a_{m1}\boldsymbol{B} & \cdots & a_{mn}\boldsymbol{B} \end{bmatrix}, \quad \text{where } \boldsymbol{A} = (a_{ij}).$$

We will be using the following important property of the Kronecker product. Consider a matrix $\boldsymbol{C} \in R^{n \times m}$ and let $\boldsymbol{c}_i \in R^n$, $i = 1, \ldots, m$ be its column vectors, so that $\boldsymbol{C} = [\boldsymbol{c}_1, \ldots, \boldsymbol{c}_m]$. We denote by $\mathsf{vec}(\boldsymbol{C})$ the column vector of dimension $mn$ formed by stacking the columns $\boldsymbol{c}_i$ of $\boldsymbol{C}$ on top of one another:

$$\mathsf{vec}(\boldsymbol{C}) = \begin{bmatrix} \boldsymbol{c}_1 \\ \vdots \\ \boldsymbol{c}_m \end{bmatrix} \in R^{mn}.$$

Now for any three matrices $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$ for which the matrix product $\boldsymbol{A} \cdot \boldsymbol{B} \cdot \boldsymbol{C}$ is defined, the following property holds [Lau04, Ch. 13]:

$$\mathsf{vec}(\boldsymbol{A} \cdot \boldsymbol{B} \cdot \boldsymbol{C}) = (\boldsymbol{C}^T \otimes \boldsymbol{A}) \cdot \mathsf{vec}(\boldsymbol{B})$$

(this follows from the fact that $\mathsf{vec}(\boldsymbol{x}\boldsymbol{y}^T) = \boldsymbol{y} \otimes \boldsymbol{x}$ for any two column vectors $\boldsymbol{x}$ and $\boldsymbol{y}$). Note that for any column vector $\boldsymbol{c}$, $\mathsf{vec}(\boldsymbol{c}) = \boldsymbol{c}$.

### 2.2 CLT13 multilinear map

We briefly recall the asymmetric CLT13 scheme; we refer to [CLT13] for a full description. The CLT13 scheme relies on the Chinese Remainder Theorem (CRT) representation. For large secret primes $p_k$'s, let $x_0 = \prod_{k=1}^{n} p_k$. We denote by $\mathsf{CRT}(a_1, a_2, \ldots, a_n)$ or $\mathsf{CRT}(a_k)_k$ the number $a \in \mathbb{Z}_{x_0}$ such that $a \equiv a_k \pmod{p_k}$ for all $k \in [n]$. The plaintext space of CLT13 scheme is $\mathbb{Z}_{g_1} \times \mathbb{Z}_{g_2} \times \cdots \times \mathbb{Z}_{g_n}$ for small secret integers $g_k$'s. An encoding of a vector $\mathbf{a} = (a_1, \ldots, a_n)$ at level set $S = \{i_0\}$ is an integer $\alpha \in \mathbb{Z}_{x_0}$ such that $\alpha = [\mathsf{CRT}(a_1 + g_1 r_1, \ldots, a_n + g_n r_n)/z_{i_0}]_{x_0}$ for small $r_k$'s, and where $z_{i_0}$ is a secret mask in $\mathbb{Z}_{x_0}$ uniformly chosen during the parameters generation procedure of the multilinear map. To support a $\kappa$-level multilinearity, $\kappa$ distinct $z_i$'s are used. We do not consider the straddling set system [BGK+14] since it is not relevant to our attacks.

Additions between encodings in the same level set can be done by modular additions in $\mathbb{Z}_{x_0}$. Multiplication between encodings can be done by modular multiplication in $\mathbb{Z}_{x_0}$, only when those encodings are in disjoint level sets, and the resulting encoding level set is the union of the input level sets. At the top level set $[\kappa]$, an encoding of zero can be tested by multiplying it by the zero-test parameter $p_{zt} = [\prod_{i=1}^{\kappa} z_i \cdot \mathsf{CRT}(p_k^* h_k g_k^{-1})_k]_{x_0}$ in $\mathbb{Z}_{x_0}$ where $p_k^* = x_0/p_k$, and comparing the result to $x_0$. If the result is small, then the encoding encodes a zero vector.[4]

[4] In this paper, for simplicity of notation, we only consider a single zero-testing element instead of a vector thereof [CLT13].

### 2.3 Indistinguishability obfuscation

We borrow the definition of indistinguishability obfuscation from [GGH+13b], where $\mathsf{iO}$ for circuits are defined.

**Definition 1 (Indistinguishability Obfuscator ($\mathsf{iO}$)).** *A uniform PPT machine $i\mathcal{O}$ is called an indistinguishability obfuscator for a circuit class $\{\mathcal{C}_\lambda\}$ if the following conditions are satisfied:*

- *For all security parameters $\lambda \in \mathbb{N}$, for all $C \in \mathcal{C}_\lambda$, for all inputs $x$, we have that*

$$Pr[C'(x) = C(x) \ : \ C' \leftarrow i\mathcal{O}(\lambda, C)] = 1.$$

- *For any (not necessarily uniform) PPT distinguisher $D$, there exists a negligible function $\alpha$ such that the following holds: For all security parameters $\lambda \in \mathbb{N}$, for all pairs of circuits $C_0, C_1 \in \mathcal{C}_\lambda$, we have that if $C_0(x) = C_1(x)$ for all inputs $x$, then*

$$|Pr[D(i\mathcal{O}(\lambda, C_0)) = 1 - Pr[D(i\mathcal{O}(\lambda, C_1)) = 1| \leq \alpha(\lambda).$$

Circuits can be directly obfuscated using circuit obfuscators [Zim15, AB15]. However, most of the $\mathsf{iO}$ candidate obfuscators (see [GGH+13b, MSW14, AGIS14], [PST14, BMSZ16, GMM+16]) first convert the circuits to matrix branching programs, randomize them, and then obfuscated them using a candidate multilinear maps scheme such as [GGH13a, CLT13, GGH15].

Obviously, for the converted branching program $\boldsymbol{B}$, the $\mathsf{iO}$ obfuscator $\mathcal{O}$ should preserve the functionality: $\boldsymbol{B}(x) = \mathcal{O}(\boldsymbol{B})(x)$ for all $x$. Moreover, for two functionally-equivalent branching programs $\boldsymbol{B}$ and $\boldsymbol{B}'$, $\mathcal{O}(\mathbf{B})$ and $\mathcal{O}(\mathbf{B}')$ should be computationally indistinguishable, unless they have different length or types of matrices. The concrete instance of such branching programs and their obfuscations are described in Section 3.1 and 3.2, respectively.

Note that, while the candidate multilinear maps [GGH13a, CLT13, GGH15] have recently been found to fail to securely realize multi-party key exchanges (see [HJ15, CHL+15, CLLT16]), few weaknesses were found in the $\mathsf{iO}$ candidates over CLT13 (and GGH15 [GGH15]), mainly due to the absence of the low-level encodings of zeroes in the public domain. In [CGH+15], Coron et al. described an attack against the circuit obfuscators for simple circuits, and the GGHRSW obfuscator for branching programs with a special decomposable structure (but not on oblivious branching programs). Annihilations attacks [MSZ16] were recently introduced and allowed to break many $\mathsf{iO}$ candidates over GGH13; however, they do not carry to obfuscators over CLT13 as far as we know.

## 3 Zeroizing attack on indistinguishability obfuscation of simple branching programs

For simplicity, we describe our attack on the simple single input branching program introduced in [MSZ16]. We will show how to generalize our attack to oblivious branching programs with arbitrary functionalities in Section 4.

### 3.1 Target branching program

We consider the following branching program $\mathbf{A}$ that evaluates to zero for all $t$-bit inputs. Let us first define the function which describes what input bit is examined at the $i$-th step:

$$\mathsf{inp}(i) = \min(i, 2t + 1 - i) \text{ for } i \in [2t]\,.$$

Now, the branching program is defined as follows:

$$\mathbf{A} = \{\mathsf{inp}, \boldsymbol{A}_0, \boldsymbol{A}_{2t+1}, \boldsymbol{A}_{i,b} \mid i \in [2t], b \in \{0,1\}\},$$

where

$$\boldsymbol{A}_0 = [0\ 1],\ \boldsymbol{A}_{2t+1} = [1\ 0]^T,\ \boldsymbol{A}_{i,0} = \boldsymbol{A}_{i,1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \text{ for } i \in [2t].$$

It is evaluated in the usual way on $x \in \{0,1\}^t$:

$$\mathbf{A}(x) := \boldsymbol{A}_0 \times \prod_{i=1}^{2t} \boldsymbol{A}_{i,x_{\mathsf{inp}(i)}} \times \boldsymbol{A}_{2t+1}.$$

### 3.2 Obfuscation of branching programs

To obfuscate a branching program, we follow the standard recipe of indistinguishability obfuscation constructions: use Killian style randomization with extra scalar multiplications by random numbers, and encode the resulting matrices with the candidate multilinear maps.

Let us describe the obfuscation procedure of the branching program $\mathbf{A}$ from Section 3.1, over the CLT13 multilinear map. Let $\prod_{k=1}^{n} \mathbb{Z}_{g_k}$ be the plaintext space of the CLT13 map, and denote $g = \prod_{k=1}^{n} g_k$. We first choose random invertible matrices $\{\boldsymbol{R}_i \in \mathbb{Z}_g^{2\times 2}\}_{i\in[2t+1]}$ and non-zero scalars $\{\alpha_{i,x} \in \mathbb{Z}_g\}_{i\in[2t],b\in\{0,1\}}$. Then the matrices in the branching program $\mathbf{A}$ are randomized using Killian randomization, and we define $\widetilde{\mathbf{A}}$ the randomized branching program:

$$\widetilde{\mathbf{A}} = \{\mathsf{inp}, \widetilde{\boldsymbol{A}}_0, \widetilde{\boldsymbol{A}}_{2t+1}, \widetilde{\boldsymbol{A}}_{i,b} \mid i \in [2t], b \in \{0,1\}\}$$

where

$$\widetilde{\boldsymbol{A}}_0 = \boldsymbol{A}_0 \cdot \boldsymbol{R}_1^{-1},\ \widetilde{\boldsymbol{A}}_{2t+1} = \boldsymbol{R}_{2t+1} \cdot \boldsymbol{A}_{2t+1},\ \widetilde{\boldsymbol{A}}_{i,b} = \alpha_{i,b} \cdot \boldsymbol{R}_i \cdot \boldsymbol{A}_{i,b} \cdot \boldsymbol{R}_{i+1}^{-1},$$

for $i \in [2t]$, $b \in \{0,1\}$.

Next, the randomized branching program $\widetilde{\mathbf{A}}$ is encoded using the CLT13 scheme. In order to evaluate the randomized branching program, our multilinear map must accommodate $\kappa = 2t+2$ products, i.e. the multilinearity level is set to $[\kappa]$. Each element $\tilde{a} \in \mathbb{Z}_g$ of the matrices $\widetilde{\boldsymbol{A}}_{i,b}$'s is considered as a vector $([\tilde{a}]_{g_1}, \ldots, [\tilde{a}]_{g_n}) \in \mathbb{Z}_{g_1} \times \cdots \times \mathbb{Z}_{g_n}$, and encoded as an integer $\hat{a} \in \mathbb{Z}_{x_0}$ at level $S = \{i\}$. In particular, we have that $\hat{a} = [\mathsf{CRT}([\tilde{a}]_{g_1} + g_1 r_1, \ldots, [\tilde{a}]_{g_n} + g_n r_n)/z_i]_{x_0}$ for small random integers $r_k$'s. The matrices $\widetilde{\boldsymbol{A}}_0$ and $\widetilde{\boldsymbol{A}}_{2t+1}$ are encoded analogously.

The resulting obfuscated branching program is

$$\widehat{\mathbf{A}} = \{\mathsf{inp}, \widehat{\boldsymbol{A}}_0, \widehat{\boldsymbol{A}}_{2t+1}, \widehat{\boldsymbol{A}}_{i,b} \mid i \in [2t], b \in \{0,1\}\}$$

where $\widehat{\boldsymbol{A}}_{i,b}$ is an entry-wise encoding of $\widetilde{\boldsymbol{A}}_{i,b}$. The obfuscated branching program $\widehat{\mathbf{A}}$ can be evaluated in the usual way: define $A(x)$ be

$$A(x) := \widehat{\boldsymbol{A}}_0 \times \prod_{i=1}^{2t} \widehat{\boldsymbol{A}}_{i,x_{\mathsf{inp}(i)}} \times \widehat{\boldsymbol{A}}_{2t+1} \times p_{zt} \bmod x_0.$$

Then $\widehat{\mathbf{A}}(x) = 0$ if and only if $A(x)$ is small compared to $x_0$.

### 3.3 Attack over CLT13 encoding

As in the previous zeroizing attacks [CHL+15, CGH+15] against the CLT13 graded encoding scheme, our approach will be to decompose the zero-tested values $A(x)$ into a product of several factors that can be made to vary independently. We then use those varying factors to construct a matrix that will reveal the factorization of the modulus $x_0$, and hence entirely break the security of the scheme.

To obtain this decomposition, we will rely on the identity $\mathsf{vec}(\boldsymbol{ABC}) = (\boldsymbol{C}^T \otimes \boldsymbol{A})\,\mathsf{vec}(\boldsymbol{B})$ (see Section 2.1). First, we define several matrices $\boldsymbol{B}(x)$, $\boldsymbol{B'}(x)$, $\boldsymbol{C}(x)$, $\boldsymbol{C'}(x)$, and $\boldsymbol{D}(x)$ as products of consecutive factors appearing in the product $A(x)$:

$$\begin{aligned} A(x) &:= \widehat{\boldsymbol{A}}_0 \times \prod_{i=1}^{2t} \widehat{\boldsymbol{A}}_{i,x_{\mathsf{inp}(i)}} \times \widehat{\boldsymbol{A}}_{2t+1} \times p_{zt} \bmod x_0 \\ &= \underbrace{\widehat{\boldsymbol{A}}_0 \cdot \prod_{i=1}^{s} \widehat{\boldsymbol{A}}_{i,x_{\mathsf{inp}(i)}}}_{\boldsymbol{B}(x)} \times \underbrace{\widehat{\boldsymbol{A}}_{s+1,x_{\mathsf{inp}(s+1)}}}_{\boldsymbol{C}(x)} \times \underbrace{\prod_{i=s+2}^{2t-s-1} \widehat{\boldsymbol{A}}_{i,x_{\mathsf{inp}(i)}}}_{\boldsymbol{D}(x)} \\ &\quad \times \underbrace{\widehat{\boldsymbol{A}}_{2t-s,x_{\mathsf{inp}(2t-s)}}}_{\boldsymbol{C'}(x)} \times \underbrace{\prod_{i=2t-s+1}^{2t} \widehat{\boldsymbol{A}}_{i,x_{\mathsf{inp}(i)}} \cdot \widehat{\boldsymbol{A}}_{2t+1}}_{\boldsymbol{B'}(x)} \times p_{zt} \bmod x_0. \end{aligned}$$

Using the identity above, we can then rewrite $A(x)$ as follows:

$$\begin{aligned} A(x) &= \boldsymbol{B}(x) \times (\boldsymbol{C}(x)\boldsymbol{D}(x)\boldsymbol{C'}(x)) \times \boldsymbol{B'}(x) \times p_{zt} \bmod x_0 \\ &= \mathsf{vec}\left(\boldsymbol{B}(x) \times \big(\boldsymbol{C}(x)\boldsymbol{D}(x)\boldsymbol{C'}(x)\big) \times \boldsymbol{B'}(x)\right) \times p_{zt} \bmod x_0 \\ &= \big(\boldsymbol{B'}(x)^T \otimes \boldsymbol{B}(x)\big) \times \mathsf{vec}\left(\boldsymbol{C}(x)\boldsymbol{D}(x)\boldsymbol{C'}(x)\right) \times p_{zt} \bmod x_0 \\ &= \big(\boldsymbol{B'}(x)^T \otimes \boldsymbol{B}(x)\big) \times \big(\boldsymbol{C'}(x)^T \otimes \boldsymbol{C}(x)\big) \times \mathsf{vec}\left(\boldsymbol{D}(x)\right) \times p_{zt} \bmod x_0. \end{aligned}$$

Furthermore, recall that CRT values have the property that $\mathsf{CRT}(p_k^* \cdot u_k)_k = \sum_k p_k^* \cdot u_k \bmod x_0$ for any tuple $(u_k)_k$, and the relation holds over $\mathbb{Z}$ when the $u_k$'s are small compared to the $p_k$'s. Now, for a multilinear encoding $\alpha$ with level set $S$, denote by $[\alpha]^{(k)}$ its underlying CRT component modulo $p_k$ (and similarly for vectors and matrices of encodings); in other words:

$$\alpha = \mathsf{CRT}\big([\alpha]^{(1)}, \ldots, [\alpha]^{(n)}\big) \cdot \prod_{i \in S} z_i^{-1} \bmod x_0.$$

With that notation and in view of the definition of $p_{zt}$, the expression of $A(x)$ can be extended further as:

$$\begin{aligned} A(x) = &\left[\ldots \left[\boldsymbol{B}(x)'^T \otimes \boldsymbol{B}(x)\right]^{(k)} \ldots\right] \\ &\times \begin{bmatrix} \ddots & & \\ & p_k^* h_k g_k^{-1} \cdot \left[\boldsymbol{C}(x)'^T \otimes \boldsymbol{C}(x)\right]^{(k)} & \\ & & \ddots \end{bmatrix} \times \begin{bmatrix} \vdots \\ \left[\mathsf{vec}(\boldsymbol{D}(x))\right]^{(k)} \\ \vdots \end{bmatrix}, \end{aligned} \tag{1}$$

where the three matrices are respectively of dimensions $1 \times 4n$, $4n \times 4n$ and $4n \times 1$. For all $x$, the fact that the branching program evaluates to zero (and hence $A(x)$ is an encoding of zero) ensures that the relation holds over $\mathbb{Q}$ and not just modulo $x_0$: indeed, it guarantees that the factor that each $p_k^*$ gets multiplied with is small modulo $p_k$.

Now the key point of the attack is that the first matrix in the relation above depends only on the first $s$ bits of the input $x$, the second matrix only on the $(s+1)$-st bit of $x$, and the third matrix on the remaining $(t-s-1)$ bits of $x$. Given integers $i, j, b$ with $0 \leq i < 2^s$, $0 \leq j < 2^{t-s-1}$ and $b \in \{0,1\}$, denote by $W_{ij}^{(b)}$ the value $A(x) \in \mathbb{Z}$ corresponding to the input $x$ whose first $s$ bits are the binary expansion of $i$, whose last $(t-s-1)$ bits are the binary expansion of $j$ and whose $(s+1)$-st bit is $b$. By the above, we can write $W_{ij}^{(b)}$ in the form:

$$W_{ij}^{(b)} = \boldsymbol{X}_i \cdot \boldsymbol{U}^{(b)} \cdot \boldsymbol{Y}_j$$

where $\boldsymbol{X}_i$ is the row vector of size $4n$, $\boldsymbol{Y}_j$ the column vector of size $4n$ and $\boldsymbol{U}^{(b)}$ the square matrix of size $4n$ that appear in Equation (1).

Assuming that $2^{\min(s,t-s-1)} \geq 4n$ (which can be achieved by taking $s = \lfloor t/2 \rfloor$ as long as $2^{t/2} \geq 8n$), we can thus form two matrices $\boldsymbol{W}^{(0)}$, $\boldsymbol{W}^{(1)}$ with any choice of $4n$ indices $i$ and $j$, and those matrices satisfy a relation of the form $\boldsymbol{W}^{(b)} = \boldsymbol{X} \cdot \boldsymbol{U}^{(b)} \cdot \boldsymbol{Y}$ with $\boldsymbol{X}$, $\boldsymbol{Y}$ square matrices of dimension $4n$ independent of $b$. The attack strategy is then similar to [CGH+15]. With high probability on the sets of indices $i$ and $j$, these matrices will be invertible over $\mathbb{Q}$, and we will have:

$$\boldsymbol{W}^{(0)}\big(\boldsymbol{W}^{(1)}\big)^{-1} = \big(\boldsymbol{X}\boldsymbol{U}^{(0)}\boldsymbol{Y}\big) \cdot \big(\boldsymbol{X}\boldsymbol{U}^{(1)}\boldsymbol{Y}\big)^{-1} = \boldsymbol{X} \cdot \boldsymbol{U}^{(0)}\big(\boldsymbol{U}^{(1)}\big)^{-1} \cdot \boldsymbol{X}^{-1}.$$

In particular, the characteristic polynomials of the matrices $\boldsymbol{W}^{(0)}\big(\boldsymbol{W}^{(1)}\big)^{-1}$ and $\boldsymbol{U}^{(0)}\big(\boldsymbol{U}^{(1)}\big)^{-1}$ are equal, and since we know the $\boldsymbol{W}^{(b)}$, we can compute that common polynomial $P$ in polynomial time, together with its factorization. Now the latter matrix is block diagonal, and satisfies:

$$\boldsymbol{U}^{(0)}\big(\boldsymbol{U}^{(1)}\big)^{-1} \equiv \begin{bmatrix} \ddots & & \\ & \boldsymbol{\Gamma} \bmod p_k & \\ & & \ddots \end{bmatrix} \pmod{x_0}$$

where $\boldsymbol{\Gamma} = \big(\boldsymbol{C}_0'^T \otimes \boldsymbol{C}_0\big) \cdot \big(\boldsymbol{C}_1'^T \otimes \boldsymbol{C}_1\big)^{-1}$ (with obvious definitions for $\boldsymbol{C}_0$, $\boldsymbol{C}_0'$, $\boldsymbol{C}_1$, $\boldsymbol{C}_1'$). Therefore, $P$ decomposes as a product of factors $P_k$, $k = 1, \ldots, n$, such that $P_k(\boldsymbol{\Gamma}) \equiv 0 \pmod{p_k}$. Moreover, as characteristic polynomials over $\mathbb{Q}$ are essentially random matrices, the polynomials $P_k$ should heuristically be irreducible with high probability, and hence occur directly in the factorization of $P$ (that assumption, which is well verified in practice, appears as Conjecture 1 in [CGH+15, Section 3.3]). This yields to the complete recovery of the $p_k$'s as $p_k = \gcd\big(x_0, P_k(\boldsymbol{\Gamma})\big)$, where the $P_k$ are the irreducible factors of $P$.

Clearly, once the $p_k$'s are found, it is straightforward to break indistinguishability obfuscation. Indeed, given any two multilinear encodings at level $\{i\}$, applying rational reconstruction to their ratio modulo $p_k$ reveals $z_i \bmod p_k$, and hence the entire $z_i$. Then, even if the $g_k$'s are kept secret, rational reconstruction again applied to $p_{zt}$ allows to recover them. This makes it possible to completely "decrypt" multilinear encodings, and hence obtain the full original randomized branching program $\widetilde{\mathbf{A}}$.

In particular, we can distinguish between the obfuscation of two branching programs whenever they are inequivalent under Kilian's randomization. This applies for example to $\mathbf{A}$ and the functionally

equivalent branching program $\mathbf{A}'$ defined in Section 3.1, since $\mathbf{A}$ satisfies $\widetilde{\boldsymbol{A}}_{1,0} = \widetilde{\boldsymbol{A}}_{1,1}$ whereas $\widetilde{\boldsymbol{A}'}_{1,0} \neq \widetilde{\boldsymbol{A}'}_{1,1}$.

### 3.4 Implementation of the attack

Since the attack relies on some heuristic assumptions regarding e.g. the irreducibility of the factors of the characteristic polynomial of $\boldsymbol{U}^{(0)}\left(\boldsymbol{U}^{(1)}\right)^{-1}$ corresponding to its block diagonal submatrices, we have written an implementation to check that these assumptions were indeed satisfied in practice. The source code in Sage [S+16] is provided in Appendix A.

Running that implementation, we have verified that we could always recover the full factorization of $x_0$ efficiently.

## 4 Generality of our attack

In the previous section, we have described a zeroizing attack that breaks CLT13-based indistinguishability obfuscation for a specific branching program (previously considered in [MSZ16]) for which no previous attack was known in the CLT13 setting. In particular, that program does not have the decomposable structure required to apply the attack of [CGH+15, Section 3.4]. In that sense, we do extend the scope of zeroizing attacks beyond the setting of [CGH+15].

However, our attack setting may seem quite special at first glance. In particular, the following aspects of our attack may seem to restrict its generality:

- we have described our attack against a somewhat simplified obfuscation construction, that yields $2 \times 2$ matrix encodings and does not include all the countermeasures against potential attacks suggested in [GGH+13b] and later papers;
- our attack appears to rely in a crucial way on the specific structure of the branching program $\mathbf{A}$ (and its $\mathsf{inp}$ function in particular) in order to achieve the partitioning necessary to apply zeroizing techniques;
- we only target a branching program for a very simple functionality (the identically zero function).

In this section, we show that all of these limitations can be overcome, so that our attack is in fact quite general:

- we can apply it to essentially all proposed (single-input) $\mathsf{iO}$ candidates instantiated over CLT13 multilinear maps, including the single-input variants of [GGH+13b, MSW14, AGIS14, PST14, BMSZ16, GMM+16];
- we can extend it to branching programs with an essentially arbitrary structure, including oblivious branching programs;
- we can mount it with programs achieving essentially arbitrary functionalities.

### 4.1 Attacking other obfuscators

The attack of Section 3 targets a somewhat simplified obfuscator that takes a branching program, randomizes it using Kilian-style random matrices together with multiplicative bundling with random scalars $\alpha_{i,x}$, and outputs multilinear encodings of the resulting randomized matrices directly. Actual candidate constructions of indistinguishability obfuscation in the literature, on the other hand, are usually more complicated, and typically involve extending the matrices in the original branching

program using dummy diagonal blocks that get canceled out when carrying out multilinear zero testing. The goal of these changes is usually to protect against classes of attacks that could exploit the particular algebraic structure of branching programs in undesirable ways—see e.g. [GMM+16] and references therein.

However, for the most part, these additional security features have no incidence on the applicability of our attack. This is because we only rely on the zero-testing of top-level multilinear encodings of zero being small—the precise algebraic structure of the matrices involved is essentially irrelevant for our purposes. This is in contrast, in particular, with Miles et al.'s annihilation attacks [MSZ16], which do exploit algebraic properties of the branching program matrices (such as low-degree polynomial relations they satisfy), and hence get thwarted by dummy submatrices used in [GGH+13b, GMM+16].

More precisely, the only difference between proposed obfuscators that matters in our attack is the dimension of the matrix encodings involved. If the obfuscated branching program $\widehat{\mathbf{A}}$ consists of $w \times w$ matrices instead of $2 \times 2$ matrices as in Section 3, $\boldsymbol{C}'(x)^T \otimes \boldsymbol{C}(x)$ is of dimension $w^2$. As a result, we need to construct matrices $\boldsymbol{W}^{(b)}$ of dimension $w^2 n$, and in particular the number $t$ of input bits should satisfy $2^{t/2} \geq 2w^2 n$.

Note that this condition is never a restriction in non-trivial cases: this is because $2^{t/2} < 2w^2 n$ implies that there is only a logarithmic number of input bits, or in other words a polynomial-size domain. But indistinguishability obfuscation for functions with a polynomial-size domain is trivial: it is equivalent to giving out the graph of the function in full, since it is a canonical (hence indistinguishable) representation, and anyone with access to an obfuscation can recover it in polynomial time.

We finish this paragraph by reviewing several candidate $i\mathcal{O}$ constructions and discussing how they fit within the argument above. This will prove Theorem 1, which we now recall.

**Theorem 1.** *Let $\mathcal{O}$ denote the single-input variant of the $i\mathcal{O}$ candidates in [GGH+13b, MSW14, AGIS14, PST14, BMSZ16, GMM+16] (over CLT13 multilinear maps). There exists a branching program $\mathbf{A}$ such that, given $\mathcal{O}(\mathbf{A})$, one can break the CLT13 multilinear maps in polynomial-time.*

**[AGIS14], [MSW14] and [BMSZ16].** The obfuscator described in Section 3.2 is essentially identical to the single-input versions of the constructions from [AGIS14], [MSW14] and [BMSZ16]. The only difference is that those papers do not directly encode matrices at singleton multilinear levels $\{i\}$, but use a more complicated level structure involving straddling sets. Since our attack relies on the honest evaluation of the obfuscated branching program, it automatically respects the multilinear level structure of any correct obfuscator. Therefore, it applies to those schemes *without any change*.

**[GGH+13b, GMM+16].** The main difference between the obfuscator proposed in [GGH+13b] and the one described in Section 3.2 is that the former extends the original branching program matrices $\boldsymbol{A}_{i,b}$ by random diagonal matrices $\boldsymbol{\Delta}_{i,b}$ of dimension $d = 2t + 5$ before applying Kilian's randomization and multilinear encoding (and the matrices $\boldsymbol{A}_{i,b}$ themselves are assumed to be of dimension 5 instead of 2, to accommodate for the original formulation of Barrington's theorem). In other words, the randomized branching program $\widetilde{\mathbf{A}}$ has the form:

$$\widetilde{\boldsymbol{A}}_{i,b} = \alpha_{i,b} \boldsymbol{R}_i \cdot \begin{bmatrix} \boldsymbol{A}_{i,b} & \\ & \boldsymbol{\Delta}_{i,b} \end{bmatrix} \cdot \boldsymbol{R}_{i+1},$$

with the bookend matrices $\widetilde{\boldsymbol{A}}_0$, $\widetilde{\boldsymbol{A}}_{2t+1}$ adapted in such a way that the condition:

$$\mathbf{A}(x) = 0 \quad \text{if and only if} \quad \widetilde{\boldsymbol{A}}_0 \cdot \prod_i \widetilde{\boldsymbol{A}}_{i,x_{\mathsf{inp}(i)}} \cdot \widetilde{\boldsymbol{A}}_{2t+1} = 0$$

continues to hold. Because that condition holds, our attack applies in exactly the same way, except again for the fact that the dimension of encoded matrices $\widetilde{\boldsymbol{A}}_{i,b}$ increases from 2 to $w = d+5 = 2t+10$. This means that the condition on $t$ becomes $2^{t/2} \geq 2(2t+10)^2 n$, which is, again, not a meaningful restriction.

Note that the attack does not depend on any property of the matrices $\boldsymbol{\Delta}_{i,b}$'s, other than the fact that correctness holds for $\mathbf{A}(x) = 0$. Therefore, the countermeasure of having a completely random $\boldsymbol{\Delta}_{i,b}$ proposed in [GMM+16] does not prevent the attack. This shows that the weak multilinear map model described in [GMM+16] does not prevent this zeroizing attack, while it seemingly provably prevents annihilitations attacks over GGH13 [MSZ16].

**[PST14].** The situation for the obfuscator of [PST14] is similar. In that scheme, the randomized branching program $\widetilde{\mathbf{A}}$ takes the form:

$$\widetilde{\boldsymbol{A}}_{i,b} = \alpha_{i,b} \boldsymbol{R}_i \cdot \begin{bmatrix} \boldsymbol{A}_{i,b} & \\ & \boldsymbol{I}_5 \end{bmatrix} \cdot \boldsymbol{R}_{i+1},$$

where $\boldsymbol{I}_5$ is simply the $5 \times 5$ identity matrix, and the original branching program matrices are also assumed to be of dimension 5. Again, our attack extends to that setting directly, the only difference being that the dimension of encoded matrices $\widetilde{\boldsymbol{A}}_{i,b}$ increases from 2 to $w = 10$. The fact that the scheme from [PST14] uses straddling sets has, again, no bearing on the applicability of our techniques.

### 4.2 Attacking branching programs with arbitrary structure

Another apparent limitation of our attack is related to the particular structure of the branching program $\mathbf{A}$, and in particular its $\mathsf{inp}$ function. Indeed, the key point of our attack is our ability to obtain a *partitioning* of the branching program, i.e. express the associated zero-test value $A(x)$ as a product of three successive factors depending on disjoint subsets of input bits. We achieved this by observing that $A(x)$ can be put in the form:

$$A(x) = \boldsymbol{B}(x) \cdot \boldsymbol{C}(x) \cdot \boldsymbol{D}(x) \cdot \boldsymbol{C}'(x) \cdot \boldsymbol{B}'(x) \times p_{zt} \bmod x_0$$

where $\boldsymbol{B}(x), \boldsymbol{B}'(x)$ depend on one subset of input bits, $\boldsymbol{C}(x)$, $\boldsymbol{C}'(x)$ a different, disjoint subset, and $\boldsymbol{D}(x)$ on a third subset disjoint from the first two. We then used the tensor product identity mentioned in Section 2.1 to reorder those matrices so as to get a factor depending only on $\boldsymbol{B}(x)$ and $\boldsymbol{B}'(x)$ on the left, another one depending only on $\boldsymbol{C}(x)$ and $\boldsymbol{C}'(x)$ in the middle, and a last one depending only on $\boldsymbol{D}(x)$ on the right:

$$A(x) = \left(\boldsymbol{B}'(x)^T \otimes \boldsymbol{B}(x)\right) \times \left(\boldsymbol{C}'(x)^T \otimes \boldsymbol{C}(x)\right) \times \mathsf{vec}\left(\boldsymbol{D}(x)\right) \times p_{zt} \bmod x_0.$$

This technique seems to rely in an essential way on the order in which input bits are assigned to successive branching program layers, and although we did not come up with the branching program $\mathbf{A}$ ourselves (as it was proposed earlier in [MSZ16]), we have to admit that it is rather special.

Indeed, proposed candidate $\mathsf{iO}$ constructions are often supposed to operate on *oblivious* branching programs, whose length is a multiple of the number $t$ of input bits and whose $\mathsf{inp}$ function is fixed to $\mathsf{inp}(i) = (i \bmod t) + 1$ (i.e. the input bits are associated to successive layers in cyclic order). This is natural, since all branching programs can be trivially converted to that form, and a canonical $\mathsf{inp}$ function is needed to ensure indistinguishability. However, the branching program $\mathbf{A}$ above is *not* oblivious, and it isn't immediately clear that our partitioning technique based on tensor products extends to that case.

Fortunately, it turns out that our technique does extend to oblivious (and hence to arbitrary) branching programs as well, at the cost of an increase in the dimension of the matrix encodings involved. There is in fact a simple greedy algorithm that will convert any scalar expression consisting of a product of three types of matrices $\boldsymbol{B}_i$, $\boldsymbol{C}_i$, $\boldsymbol{D}_i$ to an equal product of three factors, the first of which involves only the $\boldsymbol{B}_i$'s, the second only the $\boldsymbol{C}_i$'s and the third only the $\boldsymbol{D}_i$'s. Writing down a description of the algorithm would be somewhat tedious, but it is easy to understand on an example.

If we consider for example an oblivious branching program $\mathbf{A}_2$ of length $2t$ (i.e. with two groups of $t$ layers associated with all successive input bits), the corresponding zero-test value can be put in the form:

$$A(x) = \boldsymbol{B} \cdot \boldsymbol{C} \cdot \boldsymbol{D} \cdot \boldsymbol{B}' \cdot \boldsymbol{C}' \cdot \boldsymbol{D}' \cdot p_{zt} \bmod x_0$$

where, again, $\boldsymbol{B}, \boldsymbol{B}'$ depend on one subset of input bits, $\boldsymbol{C}$, $\boldsymbol{C}'$ a different, disjoint subset, and $\boldsymbol{D}$, $\boldsymbol{D}'$ on a third subset disjoint from the first two (and we omit the dependence of these matrices on $x$ to simplify notations). The matrices all have dimension $w \times w$, except the first and the last, which are of dimension $1 \times w$ and $w \times 1$ respectively. Denoting by $A_{zt}$ the value such that $A(x) = A_{zt} \cdot p_{zt} \bmod x_0$, we can then put $A_{zt}$ in the desired partitioned form as follows:

$$\begin{aligned}
A_{zt} &= \boldsymbol{BC} \cdot \mathsf{vec}\left(\boldsymbol{D} \cdot (\boldsymbol{B}'\boldsymbol{C}') \cdot \boldsymbol{D}'\right) \\
&= \boldsymbol{BC}\left(\boldsymbol{D}'^T \otimes \boldsymbol{D}\right) \mathsf{vec}(\boldsymbol{I}_w \boldsymbol{B}'\boldsymbol{C}') \\
&= \boldsymbol{BC}\left(\boldsymbol{D}'^T \otimes \boldsymbol{D}\right)\left(\boldsymbol{C}'^T \otimes \boldsymbol{I}_w\right) \mathsf{vec}(\boldsymbol{B}') \\
&= \left(\mathsf{vec}(\boldsymbol{B}')^T \otimes \boldsymbol{B}\right) \cdot \mathsf{vec}\left(\boldsymbol{C}\left(\boldsymbol{D}'^T \otimes \boldsymbol{D}\right)\left(\boldsymbol{C}'^T \otimes \boldsymbol{I}_w\right)\right) \\
&= \left(\mathsf{vec}(\boldsymbol{B}')^T \otimes \boldsymbol{B}\right) \cdot \left(\boldsymbol{C}' \otimes \boldsymbol{I}_w \otimes \boldsymbol{C}\right) \cdot \mathsf{vec}\left(\boldsymbol{D}'^T \otimes \boldsymbol{D}\right),
\end{aligned}$$

and clearly a similar procedure works for any number of layer groups, allowing us to adapt the attack to oblivious branching programs in general.

However, for an oblivious branching program of length $mt$ (with $m$ groups of $t$ layers), we can see that the dimension of the resulting square matrix in the middle is given by $w^{2m-1}$, and therefore, we need to have $2^{t/2} \geq nw^{2m-1}$ to obtain sufficiently many zeros to apply the zeroizing technique. As a result, we can attack oblivious branching programs only when the number $m$ of layer groups is not too large compared to the number $t$ of input bits. In particular, we cannot break the obfuscation of oblivious branching programs with length greater than $\omega(t^2)$ using that technique.

Thus, in principle, using oblivious branching programs whose length is quite large compared to the number of inputs might be an effective countermeasure against our attack. It remains to be seen whether further improvements could yield to a successful attack against oblivious branching programs of length $\Omega(t^c)$ for $c > 2$.

On the flip side, we will see below that by adding "dummy" input bits, we can *pad* essentially any oblivious branching program into another oblivious branching program that computes the same functionality (ignoring the dummy input bits), with the same number of layer groups, and whose obfuscation is broken using our techniques.

### 4.3 Attacking arbitrary functionalities

The attack on Section 3 was described against a branching program for the always-zero function. Since we do not use any property of the underlying matrices other than the fact that the program evaluates to zero on many inputs, it is clear that the attack should extend to branching programs for other functionalities as well. Describing the class of functionalities we can capture in that way is not easy, however.

If we take for example a branching program $\mathbf{A}''$ with the same input size, the same length and the same $\mathsf{inp}$ function as $\mathbf{A}$ (and with encoding matrices of dimension $w$, say), then a sufficient condition for the attack to apply to $\mathbf{A}''$ is essentially that we can find sufficiently many "contiguous" inputs on which the program evaluates to zero. More precisely, suppose that we can find a subset $R$ of the set $[t]$ of input bit indices and an assignment $(y_r)_{r\in R} \in \{0,1\}^R$ of these input bits such that $\mathbf{A}''$ evaluates to zero on all inputs $x \in \{0,1\}^t$ that coincide with $(y_r)$ on $R$. In other words:

$$\big(\forall r \in R,\ x_r = y_r\big) \Longrightarrow \mathbf{A}''(x) = 0.$$

Then we can break the obfuscation of $\mathbf{A}''$ using the obfuscator of Section 3.2 as soon as $2^{(t-r)/2} \geq 2w^2n$. The idea is simply to apply the attack in 3.3 with $s$ chosen in such a way that $s+1$ is exactly the $(\lfloor (t-r)/2 \rfloor + 1)$-st element of $[t] \setminus R$ (in increasing order). Then, $A(x)$ satisfies Equation (1) for all values of $x$ with $x_r = y_r$ for $r \in R$. This provides at least $2^{(t-r)/2-1}$ choices for $\boldsymbol{X}_i$, $2^{(t-r)/2-1}$ for $\boldsymbol{Y}_j$ and two choices for $\boldsymbol{U}^{(b)}$, so we have enough zero values to apply the attack.

While the condition above is quite contrived, it should be satisfied by many branching programs (especially as $t-r$ can be chosen to be logarithmic: it follows that almost all functionalities should satisfy the condition), including many natural examples (a branching program whose underlying circuit is the nontrivial conjunction of two sub-circuits, one of which depends only on $t-r$ input bits would be an example). But it gives little insight into the class of functionalities we end up capturing.

A different angle of approach towards this problem is the padding technique already considered in [MSZ16, Section 3.3]. Given a branching program $\mathbf{A}_0$ implementing any functionality and for which we can find an input where it evaluates to zero, we can convert it into another branching program $\mathbf{A}_0^*$ with slightly more input bits, that implements the same functionality (it simply ignores the additional dummy input bits and evaluates to the same values as $\mathbf{A}_0$ everywhere), and whose obfuscation is broken using our attack.

This is in fact trivial: take the branching program $\mathbf{A}_0$, and append to it (before the final bookend matrix) additional layers associated with the new input bits consisting entirely of identity matrices, in the same order as the $\mathsf{inp}$ function of the branching program $\mathbf{A}$ from Section 3.1. Since all the added layers contain only identity matrices, they do not change the functionality at all. Then, if we simply fix the non-dummy input bits to the value on which we know $\mathbf{A}_0$ vanishes, we are exactly reduced to the setting of Section 3.3, and our attack applies directly.

This may be a bit *too* trivial, however, since we could just as well append a branching program with a "decomposable" structure in the sense of [CGH+15, Section 3.4], and the corresponding attack would apply already.

A less trivial observation is that we can start from any *oblivious* branching program $\mathbf{A}_0$ (for which we know an input evaluating to zero), and convert it to another *oblivious* branching program $\mathbf{A}_0^*$ with more input bits but the same number of layer groups, that implements the same functionality in the sense above, and whose obfuscation is, again, broken using our attack.

The idea this time is to add layers associated with the dummy input bits with all-identity matrices in each layer group. This does not change the functionality, and once we fix the original input bits to the input evaluating to zero, we are reduced to breaking an oblivious branching program for the always-zero function with a fixed number $m$ of layer groups and a number of input bits that we can choose. By the discussion of Section 4.2 above, if the matrix encodings are of dimension $w$, it suffice to add $t$ dummy inputs bits where $2^{t/2} \geq nw^{2m-1}$, which is always achievable.

## 5 Conclusion

Our attack shows that the single-input candidate iO constructions for branching programs over the CLT13 multilinear map proposed in the literature should be considered insecure. We leave as a challenging open problem how to extend our attack to the dual-input iO schemes, and to GGH13.

## References


AB15. Benny Applebaum and Zvika Brakerski. Obfuscating circuits via composite-order graded encoding. In *TCC (2)*, volume 9015 of *LNCS*, pages 528–556. Springer, 2015.

AGIS14. Prabhanjan Vijendra Ananth, Divya Gupta, Yuval Ishai, and Amit Sahai. Optimizing obfuscation: Avoiding barrington's theorem. In *ACM CCS*, pages 646–658. ACM, 2014.

BGK+14. Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Protecting obfuscation against algebraic attacks. In *EUROCRYPT*, volume 8441 of *LNCS*, pages 221–238. Springer, 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 *EUROCRYPT (2)*, volume 9057 of *LNCS*, pages 563–594. Springer, 2015.

BMSZ16. 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.

CGH+15. Jean-Sébastien Coron, Craig Gentry, Shai Halevi, Tancrède Lepoint, Hemanta K. Maji, Eric Miles, Mariana Raykova, Amit Sahai, and Mehdi Tibouchi. Zeroizing without low-level zeroes: New MMAP attacks and their limitations. In Rosario Gennaro and Matthew Robshaw, editors, *CRYPTO (1)*, volume 9215 of *LNCS*, pages 247–266. Springer, 2015.

CHL+15. Jung Hee Cheon, Kyoohyung Han, Changmin Lee, Hansol Ryu, and Damien Stehlé. Cryptanalysis of the multilinear map over the integers. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT (1)*, volume 9056 of *LNCS*, pages 3–12. Springer, 2015.

CLLT16. Jean-Sébastien Coron, Moon Sung Lee, Tancrède Lepoint, and Mehdi Tibouchi. Cryptanalysis of GGH15 multilinear maps. In *CRYPTO (2)*, pages 607–628, 2016.

CLT13. 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.

GGH13a. 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.

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 *FOCS*, pages 40–49. IEEE Computer Society, 2013.

GGH15. Craig Gentry, Sergey Gorbunov, and Shai Halevi. Graph-induced multilinear maps from lattices. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC (2)*, volume 9015 of *LNCS*, pages 498–527. Springer, 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 *TCC (B)*, LNCS. Springer, 2016.

HJ15. Yupu Hu and Huiwen Jia. Cryptanalysis of GGH map. Cryptology ePrint Archive, Report 2015/301, 2015. Available at http://eprint.iacr.org/2015/301. To appear at EUROCRYPT 2016.

Lau04. Alan J. Laub. *Matrix Analysis For Scientists And Engineers*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2004.

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

MSZ16. Eric Miles, Amit Sahai, and Mark Zhandry. Annihilation attacks for multilinear maps: Cryptanalysis of indistinguishability obfuscation over GGH13. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO (2)*, volume 9815 of *LNCS*, pages 629–658. Springer, 2016.

PST14. Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation from semantically-secure multilinear encodings. In *CRYPTO (1)*, volume 8616 of *LNCS*, pages 500–517. Springer, 2014.

S+16. William Stein et al. *Sage Mathematics Software (Version 7.0)*, 2016. `http://www.sagemath.org`.

Zim15. Joe Zimmerman. How to obfuscate programs directly. In *EUROCRYPT (2)*, volume 9057 of *LNCS*, pages 439–467. Springer, 2015.

## A Source code of the attack

```
lam=20
n=3
t=ceil(log(4*n,2)+1)*2+2
l0=4*n
inp=[min(i,2*t-1-i) for i in range(2*t)]

def cmod(x,n):
  z=x%n
  if (z>n//2):
    z=z-n
  return z

def cmodM(M,n):
  Z=matrix(ZZ,M.nrows(),M.ncols())
  for i in range(M.nrows()):
    for j in range(M.ncols()):
      Z[i,j]=cmod(M[i,j],n)
  return Z

def encode(m,idx,SK,PP):
  [lam,spb,lpb,x0,pzt]=PP
  [p,g,z,H]=SK
  msg=[ZZ(m)%g[i] for i in range(n)]
  rr=[ZZ.random_element(2^spb) for i in range(n)]
  emsg=[msg[i]+g[i]*rr[i] for i in range(n)]
  tmp=crt(emsg,p)/z[idx]%x0
  return tmp

def encodem(M,idx,SK,PP):
  [lam,spb,lpb,x0,pzt]=PP
  n1=M.nrows(); n2=M.ncols()
  tmp=matrix(Integers(x0),n1,n2)
  for i in range(n1):
    for j in range(n2):
      tmp[i,j]=encode(M[i,j],idx,SK,PP)
  return tmp

def instgen(lam,n,t):
  lbr=2*t+2
  spb=lam
```

```
lpb=2*(spb+2)*lbr+spb

print "params: n, t, spb, lpb =",n,t,spb,lpb

g=[random_prime(2^spb) for i in range(n)]
q=prod(g)
Zq=Integers(q)

p=[random_prime(2^lpb) for i in range(n)]
x0=prod(p)
pis=[x0/p[i] for i in range(n)]

H=matrix(ZZ,n,n,[ZZ.random_element(2^spb) for i in range(n^2)])

Pih=[[pis[i]*H[j][i]*(g[i]^(-1)%p[i]) for i in range(n)] for j in range(n)]

z=[Integers(x0).random_element() for i in range(lbr)]
Z=prod(z)
pzt=[Z*crt(Pih[j],p)%x0 for j in range(n)]

SK=[p,g,z,H]
PP=[lam,spb,lpb,x0,pzt]

ve0=matrix(1,2,[0,1])
ve1=matrix(2,1,[1,0])

I=identity_matrix(2)
BR0=[ve0]+[I for i in range(2*t)]+[ve1]
BR1=[ve0]+[I for i in range(2*t)]+[ve1]
BR=[BR0,BR1]

k=1
R=matrix(ZZ,2,2,[0,1,1,0])

BR0p=[ve0]+[I for i in range(2*t)]+[ve1]
BR1p=[ve0]+[R for i in range(k)]+[I for i in range(2*t-2*k)]\
    +[R for i in range(k)]+[ve1]
BRp=[BR0p,BR1p]

RM=[random_matrix(Zq,2,2) for i in range(2*t+1)]
RMi=[RM[i]^(-1) for i in range(2*t+1)]
alp=[[Zq.random_element() for i in range(2*t)],\
   [Zq.random_element() for i in range(2*t)]]

RBR0=[BR0[0]*RMi[0]]+[alp[0][i-1]*RM[i-1]*BR0[i]*RMi[i] \
   for i in [1..2*t]]+[RM[-1]*BR0[-1]]
RBR1=[BR1[0]*RMi[0]]+[alp[1][i-1]*RM[i-1]*BR1[i]*RMi[i] \
   for i in [1..2*t]]+[RM[-1]*BR1[-1]]
RBR=[RBR0,RBR1]

RBR0p=[BR0p[0]*RMi[0]]+[alp[0][i-1]*RM[i-1]*BR0p[i]*RMi[i] \
   for i in [1..2*t]]+[RM[-1]*BR0p[-1]]
RBR1p=[BR1p[0]*RMi[0]]+[alp[1][i-1]*RM[i-1]*BR1p[i]*RMi[i] \
   for i in [1..2*t]]+[RM[-1]*BR1p[-1]]
RBRp=[RBR0p,RBR1p]
```

```
  eRBR=[[encodem(RBR[0][i],i,SK,PP) for i in range(lbr)],  \
        [encodem(RBR[1][i],i,SK,PP) for i in range(lbr)]]

  eRBRp=[[encodem(RBRp[0][i],i,SK,PP) for i in range(lbr)],  \
        [encodem(RBRp[1][i],i,SK,PP) for i in range(lbr)]]

  print "Randomized BP encoded."
  return [SK,PP,BR,RBR,eRBR,BRp,RBRp,eRBRp]

def fx(x,BR,inp,PP,red=0):
  [lam,spb,lpb,x0,pzt]=PP
  st=ZZ(x).binary()
  while(len(st)<t):
    st='0'+st
  res=BR[0][0]
  for i in range(2*t):
    s=ZZ(st[inp[i]])
    if red:
      res=res*BR[s][i+1]%x0
    else:
      res=res*BR[s][i+1]
  if red:
    res=res*BR[0][-1]%x0
  else:
    res=res*BR[0][-1]
  return res

def atkio(eRBR):
  [lam,spb,lpb,x0,pzt]=PP

  m=t//2
  W0=matrix(Integers(x0),l0,l0)
  W1=matrix(Integers(x0),l0,l0)
  for i in range(l0):
    for j in range(l0):
      W0[i,j]=fx(i+2^(t//2+2)*j,eRBR,inp,PP,1)[0,0]
      W1[i,j]=fx(i+2^m+2^(t//2+2)*(j),eRBR,inp,PP,1)[0,0]

  W0p=(W0*pzt[0]).change_ring(ZZ)
  W0p=cmodM(W0p,x0)
  W1p=(W1*pzt[0]).change_ring(ZZ)
  W1p=cmodM(W1p,x0)

  T1=W0p*W1p^(-1)
  T1p=T1.charpoly()
  T1pf=T1p.factor()

  rm=t-m
  tmp1=eRBR[0][rm].tensor_product(eRBR[0][-1-rm])
  tmp2=eRBR[1][rm].tensor_product(eRBR[1][-1-rm])
  tmp3=(tmp1*tmp2^(-1)).change_ring(ZZ)%x0
  res=[]
  for i in range(len(T1pf)):
    ttmp=T1pf[i][0].subs(x=tmp3)%x0
    for j1 in range(ttmp.nrows()):
      for j2 in range(ttmp.ncols()):
```

```
        tmpgcd=gcd(ttmp[j1,j2],x0)
        if tmpgcd not in res:
          res.append(tmpgcd)
  return res

[SK,PP,BR,RBR,eRBR,BRp,RBRp,eRBRp]=instgen(lam,n,t)
[lam,spb,lpb,x0,pzt]=PP

res1=atkio(eRBR)
res2=atkio(eRBRp)
tmpsk=SK[0]
res1.sort(); res2.sort(); tmpsk.sort()
if res1==res2 and res2==tmpsk:
  print "Secret␣key␣recovered"
else:
  print "Failed"
```