Cryptanalysis of Indistinguishability Obfuscations of Circuits over GGH13*†

Daniel Apon[1], Nico Döttling[‡2], Sanjam Garg[2], and Pratyay Mukherjee[4]

1 **University of Maryland, College Park, MD, USA**
dapon@cs.umd.edu
2 **University of California, Berkeley, CA, USA**
nicodoettling@berkeley.edu
3 **University of California, Berkeley, CA, USA**
sanjamg@berkeley.edu
4 **Visa Research, Palo Alto, CA, USA**
pratyay85@gmail.com

Abstract

Annihilation attacks, introduced in the work of Miles, Sahai, and Zhandry (CRYPTO 2016), are a class of polynomial-time attacks against several candidate indistinguishability obfuscation ($i\mathcal{O}$) schemes, built from Garg, Gentry, and Halevi (EUROCRYPT 2013) multilinear maps. In this work, we provide a *general* efficiently-testable property for two single-input branching programs, called *partial inequivalence,* which we show is sufficient for our variant of annihilation attacks on several obfuscation constructions based on GGH13 multilinear maps.

We give examples of pairs of natural NC^1 circuits, which – when processed via Barrington's Theorem – yield pairs of branching programs that are partially inequivalent. As a consequence we are also able to show examples of "bootstrapping circuits," (albeit somewhat artificially crafted) used to obtain obfuscations for all circuits (given an obfuscator for NC^1 circuits), in certain settings also yield partially inequivalent branching programs. Prior to our work, no attacks on any obfuscation constructions for these settings were known.

1998 ACM Subject Classification E.3 Data Encryption

Keywords and phrases Obfuscation, Multilinear Maps, Cryptanalysis.

Digital Object Identifier 10.4230/LIPIcs.ICALP.2017.38

1 Introduction

An obfuscator is a program compiler which hides all partial implementation details of a program, intuitively. This is formalized via the notion of indistinguishability obfuscation [9]: we say an obfuscator $\mathcal{O}$ is an indistinguishability obfuscator if it holds for every pair C_0, C_1 of functionally equivalent circuits (i.e. computing the same function) that $\mathcal{O}(C_0)$ and $\mathcal{O}(C_1)$

* This is the extended abstract of the full version [4] which can be found at https://eprint.iacr.org/2016/1003. Most proofs are deferred to the full version.

† Research supported in part from 2017 AFOSR YIP Award, DARPA/ARL SAFEWARE Award W911NF15C0210, AFOSR Award FA9550-15-1-0274, NSF CRII Award 1464397, and research grants by the Okawa Foundation, Visa Inc., and Center for Long-Term Cybersecurity (CLTC, UC Berkeley). The views expressed are those of the author and do not reflect the official policy or position of the funding agencies.

‡ Nico Döttling was supported by a postdoc fellowship of the German Academic Exchange Service (DAAD).

44th International Colloquium on Automata, Languages, and Programming (ICALP 2017).
Editors: Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl;
Article No. 38; pp. 38:1–38:16

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

are indistinguishable. A recent surge of results has highlighted the importance of this notion: virtually "any cryptographic task" can be achieved assuming indistinguishability obfuscation and one-way functions [34].

All known candidate constructions of indistinguishability obfuscation, e.g. [25, 8, 6], are based on multilinear-maps [24, 21, 27][1], which have been the subjects of various attacks [16, 18, 15, 29, 19]. Among them, the attacks (e.g. [24, 29]) on GGH13 [24] multilinear maps required explicit access to "low-level" encodings of zero, or differently represented low-level encodings of zero, e.g. [18]; such low-level zero-encodings do not appear naturally in obfuscation constructions. Recently Miles, Sahai, and Zhandry [32] introduced a new class of polynomial-time[2] attacks without requiring low-level zeros against several obfuscation constructions [12, 8, 3, 31, 33] and [7], when instantiated with the GGH13 multilinear maps.

More specifically, Miles et al. [32] exhibit two simple *branching programs* (and also programs padded with those) that are functionally equivalent, yet their BGKPS-obfuscations (put forward by Barak et al. in [8]) and similar constructions [12, 3, 31, 33, 7] are efficiently distinguishable.[3] However, the branching programs considered there, in particular the *all-identity* branching program, do not appear "in the wild". More specifically, obfuscation constructions for circuits first convert an NC^1 circuit into a branching program (e.g. via Barrington's transformation) that possibly results in programs with complex structures, even if one starts with simple circuits. This brings us to the following open question:

Is it possible to attack obfuscations of complex branching programs generated from NC^1 *circuits?*

1.1 Our Contributions

In this work, we are able to answer the above question affirmatively. In particular, our main contributions are:

- We first define a *general* and efficiently-testable property of two single-input[4] branching programs called *partial inequivalence* (discussed below) and demonstrate an annihilation attack against BGKPS-like obfuscations of any two (large enough) branching programs that satisfy this property.
- Next, using implementation in Sage [35] (see the full version for details on the implementation) we give explicit examples of pairs of (functionally equivalent) natural NC^1 circuits, which when processed via Barrington's Theorem yield pairs of branching programs that are partially inequivalent – and thus, attackable.
- As a consequence of the above result, we are also able to show that the "bootstrapping circuit(s)" technique used to boost $i\mathcal{O}$ for NC^1 to $i\mathcal{O}$ for $\mathsf{P/poly}$, for a certain choice of the universal circuit (albeit artificially crafted), yield partially inequivalent branching programs in a similar manner – and are, thus, also attackable.

---

[1] The work of [2] might be seen as an exception to this: Assuming the (non-explicit) existence of indistinguishability obfuscation, they provide an explicit construction of an indistinguishability obfuscator.

[2] Several subexponential-time or quantum-polynomial-time [22, 1, 17] attacks on GGH13 multilinear maps also have been considered. We do not consider these in this paper.

[3] To avoid repetitions, from now on we will refer to the obfuscation constructions of [8, 12, 3, 31, 33] by BGKPS-like constructions that use single-input branching programs.

[4] The branching programs, where any pair of matrices in the sequence depends on a single input location, are called single-input branching programs. Such branching programs naturally evolve from Barrington's transformation on circuits.

	Branching Programs	NC^1 Circuits (Barrington's)	NC^1-to-P/poly [25, 5] [11, 28]
GGHRSW[25]	⊗	○	○
BGKPS-like constructions [12, 8, 3] [33, 31, 7]	×	⊗	⊗
Obfuscations from weak multilinear maps [26, 23]	○	○	○

Figure 1 The Attack Landscape for GGH13-based Obfuscators. In all cases, the multilinear map is [24]. ○ means no attack is known. × means a prior attack is known, and we present more general attacks for this setting. ⊗ means we give the first known attack in this setting and ⊗ means a new attack is discovered concurrently to ours (namely [13]).

Our general partial inequivalence condition is broad and seems to capture a wide range of natural single-input branching programs. However, we require the program to be large enough.[5] Additionally, we need the program to output 0 on a large number of its inputs.

Finally, our new annihilation attacks are essentially based on linear system solvers and thus quite *systematic*. This is in contrast with the attacks of Miles et al. [32] which required an exhaustive search operation rendering it hard to extend their analysis for branching programs with natural structural complexity. Therefore, at a conceptual level, our work enhances the understanding of the powers and the (potential) limits of annihilation attacks.

One limitation of our technique is that they do *not* extend to so-called dual-input branching programs. We leave it as an interesting open question.

A Concurrent and Independent work

Concurrent and independent to our work,[6] Chen et al. [13] provides a polynomial time attack against the GGHRSW construction [25] based on GGH13 (and also GGH15 [27]) maps that works for so-called "input-partitioning" branching programs. Nonetheless, their attacks are not known to extend [14] for complex branching programs evolved from NC^1 circuits (e.g. via Barrington's Transformation). Hence, our work stands as the *only* work that breaks obfuscations of NC^1 circuits based on GGH13 till date.

Change in Obfuscation landscape

Given our work and the work of Chen et al. [13] the new *attack landscape against GGH13-based obfuscators* is depicted in Figure 1. We refer the reader to [2, Figure 13] for the state of the art on obfuscation constructions based on CLT13 and GGH15 multilinear maps.

[5] Note that, for our implementation we consider circuits that are quite small, only depth 3, and the resulting Barrington programs are of length 64. However, using the implementation we then "boost" the attack to a much larger NC^1 circuits that suffice for the real-world attack (discussed in the full version) to go through.

[6] The first draft of our full version [4] appeared online concurrently as their first draft [13]. At the same time another independent work [20] appeared that provided attacks against several CLT13 based obfuscators for a broader class of programs.

1.2 Technical Overview

Below, after providing some additional backgrounds on multilinear maps and known attacks, we provide an overview of our annihilation attacks.

Multilinear Maps: Abstractly

As a first approximation, one can say that a cryptographic multilinear map system encodes a value $a \in \mathbb{Z}_p$ (where p is a large prime) by using a *homomorphic* encryption scheme equipped with some additional structure. In other words, given encodings of a and b, one can perform homomorphic computations by computing encodings of $a + b$ and $a \cdot b$. Additionally, each multilinear map encoding is associated with some *level* described by a value $i \in \{1 \dots \kappa\}$ for a fixed universe parameter κ. Encodings can be added only if they are at the same level: $\mathsf{Enc}_i(a) \oplus \mathsf{Enc}_i(b) \to \mathsf{Enc}_i(a+b)$. Encodings can be multiplied: $\mathsf{Enc}_i(a) \odot \mathsf{Enc}_j(b) \to \mathsf{Enc}_{i+j}(a \cdot b)$ if $i + j \leq \kappa$ but is meaningless otherwise. We naturally extend the encoding procedure and the homomorphic operations to encode and to compute on matrices, respectively, by encoding each term of the matrix separately. Finally, the multilinear map system comes equipped with a *zero test*: an efficient procedure for testing whether the input is an encoding of 0 at level-κ. However, such zero-test procedure is not perfect as desired when instantiated with concrete candidate multilinear maps. In particular we are interested in the imperfection in GGH13 map.

An Imperfection of the GGH13 Multilinear Maps

Expanding a little on the abstraction above, a fresh multilinear map encoding of a value $a \in \mathbb{Z}_p$ at level i is obtained by first sampling a random value μ from $\mathbb{Z}_p$ and then encoding $\mathsf{Enc}_i(a + \mu \cdot p)$. Homomorphic operations can be performed just as before, except that the randomnesses from different encodings also get computed on. Specifically, $\mathsf{Enc}_i(a + \mu \cdot p) \oplus \mathsf{Enc}_i(b + \nu \cdot p)$ yields $\mathsf{Enc}_i(a + b + (\mu + \nu) \cdot p)$ and multiplication $\mathsf{Enc}_i(a + \mu \cdot p) \odot \mathsf{Enc}_j(b + \nu \cdot p)$ yields $\mathsf{Enc}_{i+j}(a \cdot b + (b \cdot \mu + a \cdot \nu + \mu \cdot \nu \cdot p) \cdot p)$ if $i + j \leq \kappa$ but is meaningless otherwise. An imperfection of the zero-test procedure is a feature characterized by two phenomena:

1. On input $\mathsf{Enc}_\kappa(0 + r \cdot p)$ the zero-test procedure additionally reveals r in a somewhat "scrambled" form.
2. For certain efficiently computable polynomials f and a collection of scrambled values $\{r_i\}$ it is efficient to check if $f(\{r_i\}) = 0 \mod p$ or not for any choice of r_i's.[7]

This imperfection has been exploited to perform attacks in prior works, such as the one by Miles et al. [32].[8]

Matrix Branching Programs

A matrix branching program of length ℓ for n-bit inputs is a sequence $BP = \left\{A_0, \left\{A_{i,0}, A_{i,1}\right\}_{i=1}^{\ell}, A_{\ell+1}\right\}$, where $A_0 \in \{0,1\}^{1\times 5}$, $A_{i,b}$'s for $i \in [\ell]$ are in $\{0,1\}^{5\times 5}$ and $A_{\ell+1} \in \{0,1\}^{5\times 1}$. Without providing details, we note that the choice of 5×5 matrices comes from Barrington's Theorem [10]. We use the notation $[n]$ to describe the set $\{1, \dots, n\}$.

[7] One can alternatively consider the scrambled values as polynomials over $\{r_i\}$ and then check if $f(\{r_i\})$ is identically zero in $\mathbb{Z}_p$.

[8] Recent works such as [26, 23], have attempted to realize obfuscation schemes secure against such imperfection and are provably secure against our attacks. We refer to them as obfuscations from weak multilinear maps (see Figure 1).

Let inp be a fixed function such that $\mathsf{inp}(i) \in [n]$ is the input bit position examined in the i^{th} step of the branching program. The function computed by this matrix branching program is

$$f_{BP}(x) = \begin{cases} 0 & \text{if } A_0 \cdot \prod_{i=1}^{\ell} A_{i,x[\mathsf{inp}(i)]} \cdot A_{\ell+1} = 0 \\ 1 & \text{if } A_0 \cdot \prod_{i=1}^{\ell} A_{i,x[\mathsf{inp}(i)]} \cdot A_{\ell+1} \neq 0 \end{cases},$$

where $x[\mathsf{inp}(i)] \in \{0,1\}$ denotes the $\mathsf{inp}(i)^{th}$ bit of x.

The branching program described above inspects one bit of the input in each step. More generally, multi-arity branching programs inspect multiple bits in each step. For example, dual-input programs inspect two bits during each step. Our strategy only works against single-input branching programs, hence we restrict ourselves to that setting.

Exploiting the Imperfection/Weakness

At a high level, obfuscation of a branching program $BP = \{A_0, \{A_{i,0}, A_{i,1}\}_{i=1}^{\ell}, A_{\ell+1}\}$ yields a collection of encodings $\{M_0, \{M_{i,0}, M_{i,1}\}_{i=1}^{\ell}, M_{\ell+1}\}$, say all of which are obtained at level-1.[9] We let $\{Z_0, \{Z_{i,0}, Z_{i,1}\}_{i=1}^{\ell}, Z_{\ell+1}\}$ denote the randomnesses used in the generation of these encodings, where each Z corresponds to a matrix of random values (analogous to r above) in $\mathbb{Z}_p$. For every input x such that $BP(x) = 0$, we have that $M_0 \odot \bigodot_{i=1}^{\ell} M_{i,x[\mathsf{inp}(i)]} \odot M_{\ell+1}$ is an encoding of 0, say of the form $\mathsf{Enc}(0 + r_x \cdot p)$ from which r_x can be learned in a scrambled form. The crucial observations of Miles et al. [32] are: (1) for every known obfuscation construction, r_x is a *program dependent* function of $\{Z_0, \{Z_{i,0}, Z_{i,1}\}_{i=1}^{\ell}, Z_{\ell+1}\}$, and (2) for a large enough $m \in \mathbb{Z}$ the values $\{r_{x_k}\}_{k=1}^{m}$ must be correlated, which in turn implies that there exists a (program-dependent) efficiently computable function f^{BP} and input choices $\{x_k^{BP}\}_{k=1}^{m}$ such that for all $k \in [m]$, $BP(x_k^{BP}) = 0$ and $f^{BP}(\{r_{x_k^{BP}}\}_{k=1}^{m}) = 0 \mod p$.[10] Further, just like Miles et al. we are interested in constructing an attacker for the *indistinguishability* notion of obfuscation. In this case, given two arbitrarily distinct programs BP and BP' (such that $\forall x, BP(x) = BP'(x)$) an attacker needs to distinguish between the obfuscations of BP and BP'. Therefore, to complete the attack, it suffices to argue that for the sequence of $\{r'_{x_k^{BP'}}\}$ values obtained from execution of BP' it holds that, $f^{BP}(\{r'_{x_k^{BP'}}\}_{k=1}^{m}) \neq 0 \mod p$. Hence, the task of attacking any obfuscation scheme reduces to the task of finding such distinguishing function f^{BP}.

Miles et al. [32] accomplishes that by presenting specific examples of branching programs, both of which implement the constant zero function, and a corresponding distinguishing function. They then extend the attack to other related branching programs that are padded with those constant-zero programs. The details of their attack [32] is quite involved, hence we jump directly to the intuition behind our envisioned more general attacks.

Partial Inequivalence of Branching Programs and Our Attacks

We start with the following observation. For BGKPS-like-obfuscations for any branching program $BP = \{A_0, \{A_{i,0}, A_{i,1}\}_{i=1}^{\ell}, A_{\ell+1}\}$ the value $s_x = r_x \mod p$ looks something like: [11]

[9] Many obfuscation constructions use more sophisticate leveling structure, typically referred to as so-called "straddling sets". However we emphasize that, this structure does not affect our attacks. Therefore we will just ignore this in our setting.

[10] This follows from the existence of an annihilating polynomial for any over-determined non-linear systems of equations. We refer to [30] for more details.

[11] Obtaining this expression requires careful analysis that is deferred to the main body of the paper. Also, by abuse of notation let $A_{0,x_{\mathsf{inp}(0)}} = A_0$, $A_{\ell+1,x_{\mathsf{inp}(\ell+1)}} = A_{\ell+1}$, $Z_{0,x_{\mathsf{inp}(0)}} = Z_0$ and $Z_{\ell+1,x_{\mathsf{inp}(\ell+1)}} = Z_{\ell+1}$.

$$s_x \simeq \prod_{i=1}^{\ell} \alpha_{i,x[\mathsf{inp}(i)]} \underbrace{\left[\sum_{i=0}^{\ell+1} \left(\prod_{j=0}^{i-1} A_{j,x_{\mathsf{inp}(j)}} \cdot Z_{i,x[\mathsf{inp}(i)]} \cdot \prod_{j=i+1}^{\ell+1} A_{j,x_{\mathsf{inp}(j)}} \right)\right]}_{t_x},$$

where $\{Z_0, \{Z_{i,0}, Z_{i,1}\}_{i=1}^{\ell}, Z_{\ell+1}\}$ where $\{Z_0, \{Z_{i,0}, Z_{i,1}\}_{i=1}^{\ell}, Z_{\ell+1}\}$ are the randomnesses contributed by the corresponding encodings. Let $\overline{x}$ denote the value obtained by flipping every bit of x (a.k.a. the bitwise complement). Now observe that the product value $\Lambda = \prod_{i=1}^{\ell} \alpha_{i,x[\mathsf{inp}(i)]} \cdot \alpha_{i,\overline{x}[\mathsf{inp}(i)]}$ is independent of x. Therefore, $u_x = s_x \cdot s_{\overline{x}} = \Lambda \cdot t_x \cdot t_{\overline{x}}$. Absorbing Λ in the $\{Z_{i,0}, Z_{i,1}\}_{i=1}^{\ell}$, we have that u_x is quadratic in the randomness values $\{Z_0, \{Z_{i,0}, Z_{i,1}\}_{i=1}^{\ell}, Z_{\ell+1}\}$, or linear in the random terms ZZ' obtained by multiplying every choice of $Z, Z' \in \{Z_0, \{Z_{i,0}, Z_{i,1}\}_{i=1}^{\ell}, Z_{\ell+1}\}$. In other words if BP evaluates to 0 both on inputs x and $\overline{x}$, the values revealed by two zero-test operations give one linear equation where the coefficients of the linear equations are program dependent. Now, if BP implements a "sufficiently non-evasive" circuit,(e.g. a PRF) such that there exist sufficiently many such inputs $x, \overline{x}$ for which $BP(x) = BP(\overline{x}) = 0$, then collecting sufficiently many values $\{x_k^{BP}, u_{x_k^{BP}}\}_{k=1}^{m}$, we get a dependent system of linear relations. Namely, there exist $\{\nu_k^{BP}\}_{k=1}^{m}$ such that $\sum_{k=1}^{m} \nu_k^{BP} \cdot u_{x_k^{BP}} = 0$. In other words, $\sum_{k=1}^{m} \nu_k^{BP} \cdot r_{x_k^{BP}} \cdot r_{\overline{x}_k^{BP}} = 0 \mod p$, where $\{\nu_k^{BP}\}_{k=1}^{m}$ depends only on the description of the branching program BP.

We remark that, in the process of linearization above we increased (by a quadratic factor) the number of random terms in the system. However, this can be always compensated by using more equations, because the number of random terms is $O(poly(n))$ (n is the input length) whereas the number of choices of input x is $2^{O(n)}$ which implies that there are exponentially many r_x available.

Note that for any branching program BP' that is "different enough" from BP, we could expect that $\sum_{k=1}^{m} \nu_k^{BP} \cdot r'_{x_k^{BP}} \cdot r'_{\overline{x}_k^{BP}} \neq 0 \mod p$ where $r'_{x_k^{BP}}$ are values revealed in executions of an obfuscation of BP'. This is because the values $\{\nu_k^{BP}\}_{k=1}^{m}$ depend on the specific implementation of BP through terms of the form $\prod_{j=0}^{i-1} A_{j,x[\mathsf{inp}(i)]}$ and $\prod_{j=i+1}^{\ell+1} A_{j,x[\mathsf{inp}(i)]}$ in s_x above. Two branching programs that differ from each other in this sense are referred to as *partially inequivalent*.[12]

What Programs are Partially Inequivalent? Attack on NC^1 circuits

The condition we put forth seems to be fairly generic and intuitively should work for large class of programs. In particular, we are interested in the programs generated from NC^1 circuits. However, due to complex structures of such programs the analysis becomes quite non-trivial.[13] Nonetheless, we manage to show via implementation in Sage [35] that the attack indeed works on a pair of branching programs obtained from a pair of simple NC^1 circuits, (say C_0, C_1) (see Sec. 6 for the circuit descriptions) by applying Barrington's Theorem. The circuits take 4 bits of inputs and on any input they evaluate to 0. In our attack we use

[12] Note that the only other constraint we need is that both BP and BP' evaluates to 0 for sufficiently many inputs, which we include in the definition (c.f. Def. 2) of partial inequivalence.

[13] Note that, the analysis of Miles et al. uses 2×2 matrices in addition to using simple branching programs. These simplifications allow them to base their analysis on many facts related to the structures of these programs. Our aim here is to see if the attack works for programs obtained from NC^1 circuits, in particular via Barrington's Theorem. So, unfortunately it is not clear if their approach can be applicable here as the structure of the programs yielded via Barrington's Theorem become much complex structurally (and also much larger in size) to analyze.

all possible 16 inputs. Furthermore, we can escalate the attack to any pair of NC^1 circuits (E_0, E_1) where $E_b = \neg C_b \wedge D_b$ $(b \in \{0,1\})$ for practically any two NC^1 circuits D_0, D_1 (we need only one input x for which $D(x) = D(\overline{x}) = 0$). We now take again a sequence of 16-inputs such that we vary the parts of all the inputs going into C_b and keep the part of inputs read by D_b fixed to x. Intuitively, since the input to D_b is always the same, each evaluation chooses the exactly same randomnesses (that is Z_i's) always. Hence in the resulting system all the random variables can be replaced by a single random variable and hence $\neg C_b \wedge D_b$ can be effectively "collapsed" to a much smaller circuit $\neg C_b \wedge 0$ (0 refers to the smallest trivial circuit consisting of only identities). Finally, again via our Sage-implementation we show that for circuits $\neg C_0 \wedge 0$ and $\neg C_1 \wedge 0$ the corresponding branching programs are partially inequivalent.

As a corollary we are also able to show examples of universal circuits U_b for which the same attack works. Since the circuit D can be almost any arbitrary NC^1 circuit, we can, in particular use any universal circuit U' and carefully combine that with C to obtain our attackable universal circuit U that results in partially inequivalent Barrington programs.

2 Notations and Preliminaries

2.1 Notations

We denote the set of natural numbers $\{1, 2, \ldots\}$ by $\mathbb{N}$, the set of all integers $\{\ldots, -1, 0, 1 \ldots\}$ by $\mathbb{Z}$ and the set of real numbers by $\mathbb{R}$. We use the notation $[n]$ to denote the set of first n natural numbers, namely $[n] \stackrel{\text{def}}{=} \{1, \ldots, n\}$.

For any bit-string $x \in \{0,1\}^n$ we let $x[i]$ denotes the i-th bit. For a matrix A we denote its i-th row by $A[i, \star]$, its j-th column by $A[\star, j]$ and the element in the i-th row and j-th column by $A[i, j]$. The i-th element of a vector $\boldsymbol{v}$ is denoted by $\boldsymbol{v}[i]$.

For more notational conventions we refer to the full version [4].

3 Attack Model for Investigating Annihilation Attacks

Similar to Miles, Sahai, and Zhandry [32] we use an abstract attack model designed to encompass the main ideas of BGKPS-like-obfuscations [8, 12, 3, 33, 31, 7] as the starting point for our attacks. We formally describe the model in the full version [4].

4 Partially Inequivalent Branching Programs

In this section, we provide a formal condition on two single-input branching programs (naturally extends to multi-input settings), namely partial inequivalence, that is sufficient for launching a distinguishing attack in the abstract model. In Section 5 we prove that this condition is sufficient for the attack.[14]

▶ **Definition 1** (Partial Products)**.** Let $\mathbf{A} = (\mathsf{inp}, A_0, \{A_{i,b}\}_{i\in[\ell],b\in\{0,1\}}, A_{\ell+1})$ be a single-input branching program of matrix-dimension d and length ℓ over n-bit input.

[14] We note that this condition is not necessary. Looking ahead, we only consider first order partially inequivalent programs in paper and remark that higher order partially inequivalent programs could also be distinguished using our techniques.

1. For any input $x \in \{0,1\}^n$ and any index $i \in [\ell+1] \cup \{0\}$ we define the vectors $\phi^{(i)}_{\mathbf{A},x}$ as follows:

$$\phi^{(i)}_{\mathbf{A},x} \stackrel{\text{def}}{=} \begin{cases} \left(A_0 \cdot \prod_{j=1}^{i-1} A_{j,x[\mathsf{inp}(j)]}\right) \otimes \left(\prod_{j=i+1}^{\ell} A_{j,x[\mathsf{inp}(j)]} \cdot A_{\ell+1}\right)^T \in \{0,1\}^{1\times d^2} & \text{if } i \in [\ell]\,, \\ \left(\prod_{j=1}^{\ell} A_{j,x[\mathsf{inp}(j)]} \cdot A_{\ell+1}\right)^T \in \{0,1\}^{1\times d} & \text{if } i = 0\,, \\ A_0 \cdot \prod_{j=1}^{\ell} A_{j,x[\mathsf{inp}(j)]} \in \{0,1\}^{1\times d} & \text{if } i = \ell+1\,. \end{cases}$$

   Additionally, define $\widetilde{\phi}^{(i)}_{\mathbf{A},x}$ for any such branching program as:

$$\widetilde{\phi}^{(i)}_{\mathbf{A},x} \stackrel{\text{def}}{=} \begin{cases} [\phi^{(i)}_{\mathbf{A},x} \mid 0^{d^2}] & \text{if } i \in [\ell] \text{ and } x[\mathsf{inp}(i)] = 0\,, \\ [0^{d^2} \mid \phi^{(i)}_{\mathbf{A},x}] & \text{if } i \in [\ell] \text{ and } x[\mathsf{inp}(i)] = 1\,, \\ \phi^{(i)}_{\mathbf{A},x} & \text{if } i = 0 \text{ or } \ell+1\,, \end{cases}$$

   where inp is a function from $[\ell] \to [n]$ and that $x[\mathsf{inp}(i)]$ denotes the bit of x corresponding to location described by $\mathsf{inp}(x)$.
2. Then the **linear partial product vector** $\boldsymbol{\phi}_{\mathbf{A},x}$ and the **quadratic partial product vector** $\boldsymbol{\psi}_{\mathbf{A},x}$ of $\mathbf{A}$ with respect to x are defined as:

$$\boldsymbol{\phi}_{\mathbf{A},x} \stackrel{\text{def}}{=} [\widetilde{\phi}^{(0)}_{\mathbf{A},x} \mid \cdots \mid \widetilde{\phi}^{(\ell+1)}_{\mathbf{A},x}] \in \{0,1\}^{1\times(2d+2^\ell d^2)}\,,$$
$$\boldsymbol{\psi}_{\mathbf{A},x} \stackrel{\text{def}}{=} \boldsymbol{\phi}_{\mathbf{A},x} \otimes \boldsymbol{\phi}_{\mathbf{A},\overline{x}} \in \{0,1\}^{1\times(2d+2^\ell d^2)^2}\,,$$

   where $\overline{x} = x \oplus 1^n$ is the compliment of x.
3. For a set of inputs $X = \{x_1, x_2, \ldots, x_m\}$ the the **linear partial product matrix** $\Phi_{\mathbf{A},X}$ and the **quadratic partial product matrix** $\Psi_{\mathbf{A},X}$ of $\mathbf{A}$ with respect to X are defined as:

$$\Phi_{\mathbf{A},X} \stackrel{\text{def}}{=} \begin{bmatrix} \boldsymbol{\phi}_{\mathbf{A},x_1} \\ \boldsymbol{\phi}_{\mathbf{A},x_2} \\ \vdots \\ \boldsymbol{\phi}_{\mathbf{A},x_m} \end{bmatrix} \in \{0,1\}^{m\times(2d+2^\ell d^2)}\,,$$

$$\Psi_{\mathbf{A},X} \stackrel{\text{def}}{=} \Phi_{\mathbf{A},X} \boxtimes \Phi_{\mathbf{A},\overline{X}} + \Phi_{\mathbf{A},\overline{X}} \boxtimes \Phi_{\mathbf{A},X} = \begin{bmatrix} \boldsymbol{\psi}_{\mathbf{A},x_1} + \boldsymbol{\psi}_{\mathbf{A},\overline{x}_1} \\ \boldsymbol{\psi}_{\mathbf{A},x_2} + \boldsymbol{\psi}_{\mathbf{A},\overline{x}_2} \\ \vdots \\ \boldsymbol{\psi}_{\mathbf{A},x_m} + \boldsymbol{\psi}_{\mathbf{A},\overline{x}_m} \end{bmatrix} \in \{0,1\}^{m\times(2d+2^\ell d^2)^2},$$

   where $\overline{X} \stackrel{\text{def}}{=} \{\overline{x}_1, \overline{x}_2, \ldots\}$.

▶ **Definition 2** (Partial Inequivalence)**.** Let $\mathbf{A}_0$ and $\mathbf{A}_1$ be two single-input matrix branching programs of matrix-dimension d and length ℓ over n-bit input. Then they are called **partially inequivalent** if there exists a polynomial in security parameter sized set X of inputs such that:

- For every $x \in X$, we have that $\mathbf{A}_0(x) = \mathbf{A}_1(x) = 0$ and $\mathbf{A}_0(\overline{x}) = \mathbf{A}_1(\overline{x}) = 0$.
- $\mathsf{colsp}\left(\Psi_{\mathbf{A}_0,X}\right) \neq \mathsf{colsp}\left(\Psi_{\mathbf{A}_1,X}\right)$.

5 Annihilation Attacks for Partially Inequivalent Programs

In this section, we describe an abstract annihilation attack against any two branching programs that are partially inequivalent. We show an attack only in the abstract model and provide details on how it can be extended to the real GGH13 setting in the full version.

▶ **Theorem 3.** *Let $\mathcal{O}$ be the generic obfuscator described in Section 3.2 of the full version. Then for any two functionally equivalent same length single-input branching programs $\mathbf{A}_0, \mathbf{A}_1$ that are partially inequivalent there exists a probabilistic polynomial time attacker that distinguishes between between $\mathcal{O}(\mathbf{A}_0)$ and $\mathcal{O}(\mathbf{A}_1)$ with noticeable probability in the abstract attack model.*

Proof.

Setup for the attack

The given branching programs $\mathbf{A}_0$ and $\mathbf{A}_1$ are provided to be functionally equivalent and partially inequivalent. Therefore there exists a set X such that: (1) for all $x \in X, \mathbf{A}_0(x) = \mathbf{A}_0(\overline{x}) = \mathbf{A}_1(x) = \mathbf{A}_1(\overline{x}) = 0$, and (2) $\mathsf{colsp}\left(\Psi_{\mathbf{A}_0,X}\right) \neq \mathsf{colsp}\left(\Psi_{\mathbf{A}_1,X}\right)$. We will assume that the adversary has access to X as auxiliary information.

Challenge

$\mathcal{A}$ receives as a challenge the obfuscation of the branching program: $\mathbf{A} \in \{\mathbf{A}_0, \mathbf{A}_1\}$ by the challenger. Recall from the description of the abstract obfuscator that, the obfuscation of program $\mathbf{A} = (\mathsf{inp}, A_0, \{A_{i,b}\}_{i\in[\ell],b\in\{0,1\}}, A_{\ell+1})$, denoted by $\mathcal{O}(\mathbf{A})$ consists of the following public variables:

$$Y_0 := A_0 \cdot R_1^{adj} + gZ_0, \quad Y_{i,b} := \alpha_{i,b} R_i \cdot A_{i,b} \cdot R_{i+1}^{adj} + gZ_{i,b}, \quad Y_0 := R_{\ell+1} \cdot A_{\ell+1} + gZ_0,$$

where the arbitrary secret variables are:

$$\widetilde{A}_0 \stackrel{\text{def}}{=} A_0 \cdot R_1^{adj}, \quad \widetilde{A}_{i,b} \stackrel{\text{def}}{=} \alpha_{i,b}(R_{i,b} \cdot A_{i,b} \cdot R_{i,b}^{adj}), \quad \widetilde{A}_{\ell+1} \stackrel{\text{def}}{=} R_{\ell+1} \cdot A_{\ell+1};$$

for random variables (i.e. Killian randomizers) $R_1, \{R_i\}, R_{\ell+1}$ and the random secret variables are denoted by $Z_0, \{Z_{i,b}\}_{i\in[\ell],b\in\{0,1\}}, Z_{\ell+1}$ and the special secret variable is g. Via change of variables we can equivalently write:

$$Y_0 := (A_0 + gZ_0) \cdot R_1^{adj}; \quad Y_{i,b} := \alpha_{i,b} R_i \cdot (A_{i,b} + gZ_{i,b}) \cdot R_{i+1}^{adj}; \quad Y_{\ell+1} := R_{\ell+1} \cdot (A_{\ell+1} + gZ_{\ell+1}).$$

Pre-Zeroizing Computation (Type-1 queries)

On receiving the obfuscation of $\mathbf{A} \in \{\mathbf{A}_0, \mathbf{A}_1\}$, $\mathcal{O}(\mathbf{A}) = \{Y_0, \{Y_{i,b}\}, Y_{\ell+1}\}$ the attacker, in the pre-zeroizing step, performs a "valid" Type-1 queries on all the inputs $X, \overline{X}$ where $X = \{x_1, \ldots, x_m\}, \overline{X} = \{\overline{x}_1, \ldots, \overline{x}_m\}$. That is, for an $x \in \{0,1\}^n$, and the abstract obfuscation $\mathcal{O}(\mathbf{A})$, the attacker queries the polynomial:

$$P_{\mathbf{A},x} = Y_0 \cdot \prod_{i=1}^{\ell} Y_{i,x[\mathsf{inp}(i)]} \cdot Y_{\ell+1}.$$

Then, expressing $P_{\mathbf{A},x}$ stratified as powers of g we obtain:

$$P_{\mathbf{A},x} = P_{\mathbf{A},x}^{(0)}(\{Y_i\}_i) + g \cdot P_{\mathbf{A},x}^{(1)}(\{Y_i\}_i) + \ldots + g^{\ell+2} \cdot P_{\mathbf{A},x}^{(\ell+2)}(\{Y_i\}_i)$$

for some polynomials $P^{(j)}_{\mathbf{A},x}(\{Y_i\}_i)$ ($j \in \{0,...,\ell+1\}$). However, by Lemma 4 we have that:

$$P^{(0)}_{\mathbf{A},x} = \rho\widehat{\alpha}_x\mathbf{A}(x)$$

for $\rho \stackrel{\text{def}}{=} \prod_i \det(R_i)$ (or $\rho I = \prod_i R_i^{adj} R_i$) and $\widehat{\alpha}_x \stackrel{\text{def}}{=} \prod_{i=1}^{\ell} \alpha_{i,x_{\mathsf{inp}(i)}}$. Since for $x \in X$ we have that $\mathbf{A}(x) = 0$, the polynomial $P^{(0)}_{\mathbf{A},x}$ is identically 0. Consequently, for each such Type 1 query the attacker receives a new handle to a variable $W_{\mathbf{A},x}$ that can be expressed as follows:

$$W_{\mathbf{A},x} = P_{\mathbf{A},x}/g = P^{(1)}_{\mathbf{A},x} + g \cdot P^{(2)}_{\mathbf{A},x} + ... + g^{\ell+1} \cdot P^{(\ell+2)}_{\mathbf{A},x}.$$

Analogously, the attacker obtains handles $W_{\mathbf{A},\overline{x}}$. After obtaining handles

$$\{(W_{\mathbf{A},x_1}, W_{\mathbf{A},\overline{x}_1}), ...(W_{\mathbf{A},x_m}, W_{\mathbf{A},\overline{x}_m})\}$$

the attacker starts the post-zeroizing phase.

Post-Zeroizing Computation

The goal of post-zeroizing computation is to *find* a polynomial Q^{ann} of degree $poly(\lambda)$ such that following holds for some $b \in \{0,1\}$:

(i) $Q^{\mathsf{ann}}(P^{(1)}_{\mathbf{A}_b,x_1}, P^{(1)}_{\mathbf{A}_b,\overline{x}_1}..., P^{(1)}_{\mathbf{A}_b,x_m}, P^{(1)}_{\mathbf{A}_b,\overline{x}_m}) \equiv 0.$

(ii) $Q^{\mathsf{ann}}(P^{(1)}_{\mathbf{A}_{1-b},x_1}, P^{(1)}_{\mathbf{A}_{1-b},\overline{x}_1}..., P^{(1)}_{\mathbf{A}_{1-b},x_m}, P^{(1)}_{\mathbf{A}_{1-b},\overline{x}_m}) \not\equiv 0.$

Clearly, this leads to an attack on the obfuscation security as $\mathcal{A}$ would receive 0 from the challenger if and only if $Q^{\mathsf{ann}}(P^{(1)}_{\mathbf{A},x_1}, P^{(1)}_{\mathbf{A},\overline{x}_1}..., P^{(1)}_{\mathbf{A},x_m}, P^{(1)}_{\mathbf{A},\overline{x}_m})$ is identically zero, hence it would receive 0 if and only if $\mathbf{A}_b$ is chosen by the challenger in the challenge phase. To find such Q^{ann} the attacker continues as follows. Observe that by Lemma 4, for every $x \in X$ we have that:

$$P^{(1)}_{\mathbf{A},x} = \rho\widehat{\alpha}_x(\boldsymbol{\phi}_{\mathbf{A},x} \cdot \boldsymbol{z}^T)', \tag{1}$$

$$P^{(1)}_{\mathbf{A},\overline{x}} = \rho\widehat{\alpha}_{\overline{x}}(\boldsymbol{\phi}_{\mathbf{A},\overline{x}} \cdot \boldsymbol{z}^T). \tag{2}$$

Next, multiplying the polynomials $P^{(1)}_{\mathbf{A},x}$ and $P^{(1)}_{\mathbf{A},\overline{x}}$ (Eq. 1 and Eq. 2) we get:

$$\widetilde{P}^{(1)}_{\mathbf{A},x} \stackrel{\text{def}}{=} P^{(1)}_{\mathbf{A},x}P^{(1)}_{\mathbf{A},\overline{x}} = \rho^2\widehat{\alpha}\left((\boldsymbol{\phi}_{\mathbf{A},x} \cdot \boldsymbol{z}) \otimes (\boldsymbol{\phi}_{\mathbf{A},\overline{x}} \cdot \boldsymbol{z})\right) \tag{3}$$

$$= \rho^2\widehat{\alpha}\left((\boldsymbol{\phi}_{\mathbf{A},x} \otimes \boldsymbol{\phi}_{\mathbf{A},\overline{x}}) \cdot (\boldsymbol{z}^T \otimes \boldsymbol{z}^T)\right) \tag{4}$$

$$= \rho^2\widehat{\alpha}(\boldsymbol{\psi}_{\mathbf{A},x} \cdot \boldsymbol{z}^T \otimes \boldsymbol{z}^T).$$

where $\widehat{\alpha} \stackrel{\text{def}}{=} \widehat{\alpha}_x\widehat{\alpha}_{\overline{x}}$ is now independent of input x.[15] Similarly we can also have:

$$\widetilde{P}^{(1)}_{\mathbf{A},\overline{x}} \stackrel{\text{def}}{=} P^{(1)}_{\mathbf{A},\overline{x}}P^{(1)}_{\mathbf{A},x} = \rho^2\widehat{\alpha}\left((\boldsymbol{\phi}_{\mathbf{A},\overline{x}} \cdot \boldsymbol{z}) \otimes (\boldsymbol{\phi}_{\mathbf{A},x} \cdot \boldsymbol{z})\right)$$

$$= \rho^2\widehat{\alpha}\left((\boldsymbol{\phi}_{\mathbf{A},\overline{x}} \otimes \boldsymbol{\phi}_{\mathbf{A},x}) \cdot (\boldsymbol{z}^T \otimes \boldsymbol{z}^T)\right)$$

$$= \rho^2\widehat{\alpha}(\boldsymbol{\psi}_{\mathbf{A},\overline{x}} \cdot \boldsymbol{z}^T \otimes \boldsymbol{z}^T).$$

[15] Here, we use the fact that the branching programs are single-input. For multi-input programs we do *not* know how to make $\widehat{\alpha}$ independent of x. The rest of the analysis does not require the programs to be single-input.

However, since field multiplication is commutative, adding we get:

$$\begin{aligned}\widetilde{P}^{(1)}_{\mathbf{A},x} + \widetilde{P}^{(1)}_{\mathbf{A},\overline{x}} = 2P^{(1)}_{\mathbf{A},x}P^{(1)}_{\mathbf{A},\overline{x}} &= \rho^2\widehat{\alpha}(\boldsymbol{\psi}_{\mathbf{A},x}\cdot \boldsymbol{z}^T \otimes \boldsymbol{z}^T) + \rho^2\widehat{\alpha}(\boldsymbol{\psi}_{\mathbf{A},\overline{x}}\cdot \boldsymbol{z}^T \otimes \boldsymbol{z}^T)\\ &= \rho^2\widehat{\alpha}(\boldsymbol{\psi}_{\mathbf{A},x} + \boldsymbol{\psi}_{\mathbf{A},\overline{x}})\cdot(\boldsymbol{z}^T \otimes \boldsymbol{z}^T)\,.\end{aligned}$$

Using the given conditions that $\Psi_{\mathbf{A}_0,X}$ and $\Psi_{\mathbf{A}_1,X}$ have distinct column spaces (and hence distinct left-kernel) the attacker can efficiently compute (e.g. via Gaussian Elimination) a vector $\boldsymbol{v}_{\mathsf{ann}} \in \{0,1\}^{1\times m}$ that belongs to its left-kernel, call it the *annihilating vector*, such that for some $b \in \{0,1\}$ we have:

$$\boldsymbol{v}_{\mathsf{ann}}\cdot\Psi_{\mathbf{A}_b,X} = 0 \qquad \text{but} \qquad \boldsymbol{v}_{\mathsf{ann}}\cdot\Psi_{\mathbf{A}_{1-b},X} \neq 0.$$

The corresponding annihilation polynomial Q^{ann} can be written as:

$$Q^{\mathsf{ann}}_{\boldsymbol{v}_{\mathsf{ann}}}(W_{\mathbf{A},x_1}, W_{\mathbf{A},\overline{x}_1}, \ldots, W_{\mathbf{A},x_m}, W_{\mathbf{A},\overline{x}_m}) = \boldsymbol{v}_{\mathsf{ann}}\cdot\begin{bmatrix} W_{\mathbf{A},x_1}W_{\mathbf{A},\overline{x}_1}\\ \vdots \\ W_{\mathbf{A},x_m}W_{\mathbf{A},\overline{x}_m}\end{bmatrix}.$$

Observe that the coefficient of g^0 in the expression $Q^{\mathsf{ann}}_{\boldsymbol{v}_{\mathsf{ann}}}(W_{\mathbf{A},x_1}, W_{\mathbf{A},\overline{x}_1}, \ldots, W_{\mathbf{A},x_m}, W_{\mathbf{A},\overline{x}_m})$ from above is equal to $Q^{\mathsf{ann}}_{\boldsymbol{v}_{\mathsf{ann}}}(P^{(1)}_{\mathbf{A}_b,x_1}, P^{(1)}_{\mathbf{A}_b,\overline{x}_1}..., P^{(1)}_{\mathbf{A}_b,x_m}, P^{(1)}_{\mathbf{A}_b,\overline{x}_m})$. Moreover this value for $\mathbf{A} = \mathbf{A}_b$ is:

$$Q^{\mathsf{ann}}_{\boldsymbol{v}_{\mathsf{ann}}}(P^{(1)}_{\mathbf{A}_b,x_1}, P^{(1)}_{\mathbf{A}_b,\overline{x}_1}..., P^{(1)}_{\mathbf{A}_b,x_m}, P^{(1)}_{\mathbf{A}_b,\overline{x}_m}) = \boldsymbol{v}_{\mathsf{ann}}\cdot\frac{\Psi_{\mathbf{A}_b,X}}{2}\cdot(\boldsymbol{z}\otimes\boldsymbol{z})^T \equiv 0$$

but for $\mathbf{A}_{1-b}$:

$$Q^{\mathsf{ann}}_{\boldsymbol{v}_{\mathsf{ann}}}(P^{(1)}_{\mathbf{A}_{1-b},x_1}, P^{(1)}_{\mathbf{A}_{1-b},\overline{x}_1}..., P^{(1)}_{\mathbf{A}_{1-b},x_m}, P^{(1)}_{\mathbf{A}_{1-b},\overline{x}_m}) = \boldsymbol{v}_{\mathsf{ann}}\cdot\frac{\Psi_{\mathbf{A}_{1-b},X}}{2}\cdot(\boldsymbol{z}\otimes\boldsymbol{z})^T \not\equiv 0.$$

Hence, the response to Type 2 query is sufficient to distinguish between obfuscation of $\mathbf{A}_b$ and $\mathbf{A}_{1-b}$ in the abstract model. This concludes the proof. ◀

Evaluations of $P^{(0)}_{\mathbf{A},\boldsymbol{x}}$ and $P^{(1)}_{\mathbf{A},\boldsymbol{x}}$

Below we state a lemma without proof (that is deferred to the full version) that described what the terms $P^{(0)}_{\mathbf{A},x}$ and $P^{(1)}_{\mathbf{A},x}$ look like.

▶ **Lemma 4.** *For every $x \in \{0,1\}^n$, we have that:*

$$P^{(0)}_{\mathbf{A},x} = \rho\widehat{\alpha}_x\mathbf{A}(x)\,,$$
$$P^{(1)}_{\mathbf{A},x} = \rho\widehat{\alpha}_x(\boldsymbol{\phi}_{\mathbf{A},x}\cdot\boldsymbol{z}^T),$$

where $\rho \stackrel{\text{def}}{=} \prod_i \det(R_i)$ and $\widehat{\alpha}_x \stackrel{\text{def}}{=} \prod_{i=1}^{\ell}\alpha_{i,x_{\mathsf{inp}(i)}}$ and $\boldsymbol{z}$ is a vector consisting of the random terms $Z_0, Z_{i,b}$, and $Z_{\ell+1}$ used to generate the obfuscation terms $Y_0, Y_{i,b}$, and $Y_{\ell+1}$ in an appropriate sequence.

Extending the Abstract Attack to GGH13 Multilinear Maps

Based on the ideas from Miles et al. we can extend our abstract attacks to actual instantiations with GGH13, that we defer to the full version [4].

6 Example of Partially Inequivalent Circuits

In this section, we show examples of pairs of NC^1 circuits such that the corresponding Barrington-implemented[16] branching programs are partially inequivalent and therefore are subject to the abstract annihilation attacks shown in Section 5. Note that here we extend the notion of partial inequivalence from branching programs to circuits in a natural way. Unless otherwise mentioned, partial inequivalence of circuits specifically imply that the corresponding branching programs generated via applying Barrington's Theorem are partially inequivalent.

6.1 Simple Pairs of Circuits that are Partially Inequivalent

Consider the following pair of circuits (C_0, C_1) each of which implements a boolean function $\{0,1\}^4 \to \{0,1\}$:

$$C_0(x) \stackrel{\text{def}}{=} (x[1] \wedge 1) \bigwedge (x[2] \wedge 0) \bigwedge (x[3] \wedge 1) \bigwedge (x[4] \wedge 0),$$

$$C_1(x) \stackrel{\text{def}}{=} (x[1] \wedge 0) \bigwedge (x[2] \wedge 0) \bigwedge (x[3] \wedge 0) \bigwedge (x[4] \wedge 0).$$

Define the set $X \stackrel{\text{def}}{=} \{0,1\}^4$. Now, we provide an implementation (see the full version [4] for more details on the implementation) in Sage [35] that evaluates the column spaces of matrices produced via applying a Barrington-implementation to the above circuits. The outcome from the implementation led us to conclude the following claim:

▶ **Claim 5.** *Let $\mathbf{A}_{C_0}, \mathbf{A}_{C_1}$ be the Barrington-Implementation of the circuits C_0, C_1 respectively, then we have that:* $\mathsf{colsp}\left(\Psi_{\mathbf{A}_{C_0},X}\right) \neq \mathsf{colsp}\left(\Psi_{\mathbf{A}_{C_1},X}\right)$.

▶ Remark. We emphasize that we use branching programs generated with a particular Barrington-implementation that makes a set of specific choices. We refer the reader to the full version [4] for the details of our implementation. Throughout this section we refer to this particular Barrington-implementation.

The circuits presented above are of constant size. Looking ahead, though, they are partially inequivalent and hence (by Theorem 3) are susceptible to the abstract attack that does not translate to a real-world attack in GGH13 setting immediately. For that we need to consider larger (albeit NC^1) circuits which we construct next based on the above circuits.

6.2 Larger Pairs of Circuits that are Partially Inequivalent

Consider *any* pair of functionally equivalent NC^1 circuits (D_0, D_1) and an input $x^\star \in \{0,1\}^n$ such that $D_0(x^\star) = D_1(x^\star) = D_0(\overline{x^\star}) = D_1(\overline{x^\star}) = 0$. Now define the circuits E_0, E_1 each of which computes a boolean function $\{0,1\}^{n+4} \to \{0,1\}$ as follows:

$$E_0(y) \stackrel{\text{def}}{=} \neg C_0(x) \wedge D_0(x'),$$

$$E_1(y) \stackrel{\text{def}}{=} \neg C_1(x) \wedge D_1(x'),$$

($\neg C$ is the circuit C with output negated) such that for each $y \in \{0,1\}^{n+4}$ we have $y = x \circ x'$ ($\circ$ denotes concatenation) where $x \in \{0,1\}^4$ and $x' \in \{0,1\}^n$. Define the input-sequence $Y \stackrel{\text{def}}{=} \{x \circ x^\star \mid x \in \{0,1\}^4\}$ (consisting of 16 inputs). Then we show the following statement.

[16] Recall that by Barrington-implementation of a circuit we mean the single-input branching program produced as a result of Barrington Theorem on the circuit. Also we implicitly assume that the branching programs are input-oblivious.

▶ **Lemma 6.** *Let* $\mathbf{A}_{E_0}, \mathbf{A}_{E_1}$ *be the Barrington-implementations of* E_0, E_1 *respectively, then we have that:* $\mathsf{colsp}\left(\Psi_{\mathbf{A}_{E_0},Y}\right) \neq \mathsf{colsp}\left(\Psi_{\mathbf{A}_{E_1},Y}\right)$.

6.3 Partially Inequivalent Universal Circuits

In this section we present constructions of (NC^1) universal circuits that, when compiled with two arbitrary distinct (NC^1) but functionally equivalent circuits as inputs, then the obfuscations of the Barrington-implementation of the compiled circuits are distinguishable by the abstract attack.

▶ **Theorem 7.** *There exists a family of* NC^1 *universal circuits* $\mathcal{U} = \{U_1, U_2, \ldots, U_v\}$ *of size* $v = O(poly(\lambda))$ *such that: given two arbitrary functionally equivalent* NC^1 *circuits* G_0, G_1 *that computes arbitrary boolean function* $\{0,1\}^n \to \{0,1\}$ *satisfying (i)* $|G_0| = |G_1| = v$ *and (ii) there exists an input* $x^\star$ *such that* $G_0(x^\star) = G_1(x^\star) = G_0(\overline{x^\star}) = G_1(\overline{x^\star}) = 0$*; then for at least one* $i \in [v]$ *the Barrington-implementations of the circuits* $U_i[G_0]$ *and* $U_i[G_1]$ *are partially inequivalent.*

References

1. 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, *Advances in Cryptology – CRYPTO 2016, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 153–178, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-53018-4_6.
2. Prabhanjan Ananth, Aayush Jain, Moni Naor, Amit Sahai, and Eylon Yogev. Universal constructions and robust combiners for indistinguishability obfuscation and witness encryption. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 491–520, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-53008-5_17.
3. 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: 21st Conference on Computer and Communications Security*, pages 646–658, Scottsdale, AZ, USA, November 3–7, 2014. ACM Press.
4. Daniel Apon, Nico Döttling, Sanjam Garg, and Pratyay Mukherjee. Cryptanalysis of indistinguishability obfuscations of circuits over ggh13. Cryptology ePrint Archive, Report 2016/1003, 2016. URL: http://eprint.iacr.org/2016/1003.
5. Benny Applebaum. Bootstrapping obfuscators via fast pseudorandom functions. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014, Part II*, volume 8874 of *Lecture Notes in Computer Science*, pages 162–172, Kaoshiung, Taiwan, R.O.C., December 7–11, 2014. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-45608-8_9.
6. Benny Applebaum and Zvika Brakerski. Obfuscating circuits via composite-order graded encoding. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015: 12th Theory of Cryptography Conference, Part II*, volume 9015 of *Lecture Notes in Computer Science*, pages 528–556, Warsaw, Poland, March 23–25, 2015. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-46497-7_21.
7. Saikrishna Badrinarayanan, Eric Miles, Amit Sahai, and Mark Zhandry. Post-zeroizing obfuscation: New mathematical tools, and the case of evasive circuits. In *Advances in Cryptology – EUROCRYPT*, 2016.

8 Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Protecting obfuscation against algebraic attacks. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 221–238, Copenhagen, Denmark, May 11–15, 2014. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-55220-5_13.

9 Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 1–18, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Heidelberg, Germany.

10 David A. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in nc^1. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing, May 28-30, 1986, Berkeley, California, USA*, pages 1–5, 1986. doi:10.1145/12130.12131.

11 Nir Bitansky, Sanjam Garg, Huijia Lin, Rafael Pass, and Sidharth Telang. Succinct randomized encodings and their applications. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th Annual ACM Symposium on Theory of Computing*, pages 439–448, Portland, OR, USA, June 14–17, 2015. ACM Press.

12 Zvika Brakerski and Guy N. Rothblum. Virtual black-box obfuscation for all circuits via generic graded encoding. In Yehuda Lindell, editor, *TCC 2014: 11th Theory of Cryptography Conference*, volume 8349 of *Lecture Notes in Computer Science*, pages 1–25, San Diego, CA, USA, February 24–26, 2014. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-54242-8_1.

13 Yilei Chen, Craig Gentry, and Shai Halevi. Cryptanalyses of candidate branching program obfuscators. Cryptology ePrint Archive, Report 2016/998, To appear in EUROCRYPT 2017, 2016. URL: http://eprint.iacr.org/2016/998.

14 Yilei Chen, Craig Gentry, and Shai Halevi. Cryptanalyses of candidate branching program obfuscators. Personal Communication, 2016.

15 Jung Hee Cheon, Pierre-Alain Fouque, Changmin Lee, Brice Minaud, and Hansol Ryu. Cryptanalysis of the new CLT multilinear map over the integers. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part I*, volume 9665 of *Lecture Notes in Computer Science*, pages 509–536, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-49890-3_20.

16 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, *Advances in Cryptology – EUROCRYPT 2015, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 3–12, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-46800-5_1.

17 Jung Hee Cheon, Jinhyuck Jeong, and Changmin Lee. An algorithm for NTRU problems and cryptanalysis of the GGH multilinear map without an encoding of zero. *IACR Cryptology ePrint Archive*, 2016:139, 2016. URL: http://eprint.iacr.org/2016/139.

18 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 *Advances in Cryptology – CRYPTO 2015 – 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, pages 247–266, 2015.

19 Jean-Sébastien Coron, Moon Sung Lee, Tancrède Lepoint, and Mehdi Tibouchi. Cryptanalysis of GGH15 multilinear maps. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part II*, volume 9815 of *Lecture Notes in Com-*

puter Science, pages 607–628, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-53008-5_21.

20 Jean-Sébastien Coron, Moon Sung Lee, Tancrède Lepoint, and Mehdi Tibouchi. Zeroizing attacks on indistinguishability obfuscation over CLT13. Cryptology ePrint Archive, Report 2016/1011, To appear in PKC 2017, 2016. URL: http://eprint.iacr.org/2016/1011.

21 Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Practical multilinear maps over the integers. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 476–493, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-40041-4_26.

22 Ronald Cramer, Léo Ducas, Chris Peikert, and Oded Regev. Recovering short generators of principal ideals in cyclotomic rings. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 559–585, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-49896-5_20.

23 Nico Döttling, Sanjam Garg, Divya Gupta, Peihan Miao, and Pratyay Mukherjee. Obfuscation from low noise multilinear maps. Cryptology ePrint Archive, Report 2016/599, 2016. http://eprint.iacr.org/2016/599.

24 Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 1–17, Athens, Greece, May 26–30, 2013. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-38348-9_1.

25 Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th Annual Symposium on Foundations of Computer Science*, pages 40–49, Berkeley, CA, USA, October 26–29, 2013. IEEE Computer Society Press.

26 Sanjam Garg, Eric Miles, Pratyay Mukherjee, Amit Sahai, Akshayaram Srinivasan, and Mark Zhandry. Secure obfuscation in a weak multilinear map model. In *TCC 2016-B*, 2016.

27 Craig Gentry, Sergey Gorbunov, and Shai Halevi. Graph-induced multilinear maps from lattices. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015: 12th Theory of Cryptography Conference, Part II*, volume 9015 of *Lecture Notes in Computer Science*, pages 498–527, Warsaw, Poland, March 23–25, 2015. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-46497-7_20.

28 Vipul Goyal, Yuval Ishai, Amit Sahai, Ramarathnam Venkatesan, and Akshay Wadia. Founding cryptography on tamper-proof hardware tokens. In Daniele Micciancio, editor, *TCC 2010: 7th Theory of Cryptography Conference*, volume 5978 of *Lecture Notes in Computer Science*, pages 308–326, Zurich, Switzerland, February 9–11, 2010. Springer, Heidelberg, Germany.

29 Yupu Hu and Huiwen Jia. Cryptanalysis of GGH map. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part I*, volume 9665 of *Lecture Notes in Computer Science*, pages 537–565, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-49890-3_21.

30 N. Kayal. The complexity of the annihilating polynomial. In *Computational Complexity, 2009. CCC'09. 24th Annual IEEE Conference on*, pages 184–193, July 2009. doi:10.1109/CCC.2009.37.

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

32 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, *Advances in Cryptology – CRYPTO 2016, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 629–658, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-53008-5_22.

33 Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation from semantically-secure multilinear encodings. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 500–517, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-44371-2_28.

34 Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *46th Annual ACM Symposium on Theory of Computing*, pages 475–484, New York, NY, USA, May 31 – June 3, 2014. ACM Press.

35 W. A. Stein et al. *Sage Mathematics Software (Version 7.3)*. The Sage Development Team, 2016. URL: http://www.sagemath.org.