Cryptanalysis of Indistinguishability Obfuscations of Circuits over GGH13

Daniel Apon, Nico Döttling, Sanjam Garg, and Pratyay Mukherjee

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 (iO) 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.

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 \(O\) 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 \(O(C_0)\) and \(O(C_1)\)
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], 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 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, 8, 3, 31, 33, 7] are efficiently distinguishable. 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 $\text{NC}^1$ 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 $\text{NC}^1$ 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 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 $\text{NC}^1$ 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 $iO$ for $\text{NC}^1$ to $iO$ for $\text{P/poly}$, 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.
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. 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, 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 NC1 circuits (e.g. via Barrington’s Transformation). Hence, our work stands as the only work that breaks obfuscations of NC1 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 NC1 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 \( i \in \{1, \ldots , \kappa \} \) for a fixed universe parameter \( \kappa \). Encodings can be added only if they are at the same level: \( \text{Enc}_i(a) \oplus \text{Enc}_i(b) \rightarrow \text{Enc}_i(a + b) \). Encodings can be multiplied: \( \text{Enc}_i(a) \odot \text{Enc}_j(b) \rightarrow \text{Enc}_{i+j}(a \cdot b) \) 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-\( \kappa \). 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 \( \mu \) from \( \mathbb{Z}_p \) and then encoding \( \text{Enc}_i(a + \mu \cdot p) \). Homomorphic operations can be performed just as before, except that the randomnesses from different encodings also get computed on. Specifically, \( \text{Enc}_i(a + \mu \cdot p) \oplus \text{Enc}_j(b + \nu \cdot p) \) yields \( \text{Enc}_i(a + b + (\mu + \nu) \cdot p) \) and multiplication \( \text{Enc}_i(a + \mu \cdot p) \odot \text{Enc}_j(b + \nu \cdot p) \) yields \( \text{Enc}_{i+j}(a \cdot b + (b \cdot \mu + a \cdot \nu + \mu \cdot \nu) \cdot p) \) 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 \( \text{Enc}_i(0 + r \cdot p) \) 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. \(^3\).\(^8\)

Multilinear Programs

A matrix branching program of length \( \ell \) for \( n \)-bit inputs is a sequence
\[
BP = \left\{ A_0, \{A_{i,0}, A_{i,1}\}_{i=1}^{\ell} \right\},
\]
where \( A_0 \in \{0, 1\}^{5 \times 5} \), \( A_{i,0} \)'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 \( 5 \times 5 \) matrices comes from Barrington’s Theorem \(^10\). We use the notation \([n]\) to describe the set \( \{1, \ldots , n\} \).

---

\(^7\) 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 inp(i) ∈ [n] is the input bit position examined in the ith step of the branching program. The function computed by this matrix branching program is

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

where \( x[i \text{inp}(i)] \in \{0,1\} \) denotes the 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 \( BP = \{A_0, \{A_{i,0}, A_{i,1}\}_{i=1}^{m}, A_{\ell+1}\} \) yields a collection of encodings \( \{M_0, \{M_{i,0}, M_{i,1}\}_{i=1}^{r}, M_{\ell+1}\} \), say all of which are obtained at level-1.\(^9\)

We let \( \{Z_0, \{Z_{i,0}, Z_{i,1}\}_{i=1}^{r}, Z_{\ell+1}\} \) 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 \( M_0 \odot \bigodot_{i=1}^{r} M_{i,x[i \text{inp}(i)]} \odot M_{\ell+1} \) is an encoding of 0, say of the form \( \text{Enc}(0+r_z \cdot p) \) from which \( r_z \) can be learned in a scrambled form.

The crucial observations of Miles et al. [32] are: (1) for every known obfuscation construction, \( r_z \) is a program dependent function of \( \{Z_0, \{Z_{i,0}, Z_{i,1}\}_{i=1}^{r}, Z_{\ell+1}\} \), and (2) for a large enough \( m \in \mathbb{Z} \) the values \( \{r_{z,k}\}_{k=1}^{m} \) must be correlated, which in turn implies that there exists a (program-dependent) efficiently computable function \( f^{BP} \) and input choices \( \{x_{BP}^{k}\}_{k=1}^{m} \) such that for all \( k \in [m] \), \( BP(x_{BP}^{k}) = 0 \) and \( f^{BP}(\{r_{z,k}\}_{k=1}^{m}) = 0 \mod p \).\(^10\) 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 \( \{r'_{x,BP}\} \) values obtained from execution of \( BP' \) it holds that, \( f^{BP}(\{r'_{x,BP}\}_{k=1}^{m}) = 0 \mod p \). 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 \( BP = \{A_0, \{A_{i,0}, A_{i,1}\}_{i=1}^{m}, A_{\ell+1}\} \) 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 \( A_{0,x_{\text{inp}}(i)} = A_0, A_{\ell+1,x_{\text{inp}}(i)} = A_{\ell+1}, Z_0,x_{\text{inp}}(i) = Z_0 \) and \( Z_{\ell+1,x_{\text{inp}}(i)} = Z_{\ell+1} \).
\[ s_x \simeq \prod_{i=1}^{\ell} \alpha_{i,x[\text{inp}(i)]} \left( \sum_{i=0}^{\ell-1} \left( \prod_{j=0}^{i-1} A_{j,x[\text{inp}(j)]} \cdot Z_{i,x[\text{inp}(i)]} \cdot \prod_{j=i+1}^{\ell} A_{j,x[\text{inp}(j)]} \right) \right), \]

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 \( \mathcal{T} \) denote the value obtained by flipping every bit of \( x \) (a.k.a. the bitwise complement). Now observe that the product value \( \Lambda = \prod_{i=1}^{\ell} \alpha_{i,x[\text{inp}(i)]} \alpha_{i,\overline{x}[\text{inp}(i)]} \) is independent of \( x \). Therefore, \( u_x = s_x \cdot s_\mathcal{T} = \Lambda \cdot t_x \cdot t_\mathcal{T} \).

Absorbing \( \Lambda \) in the \( \{Z_0, \{Z_{i,0}, Z_{i,1}\}_{i=1}^{\ell}, Z_{\ell+1} \} \), 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 \( \mathcal{T} \), 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 \( x, \mathcal{T} \) for which \( BP(x) = BP(\mathcal{T}) = 0 \), then collecting sufficiently many values \( \{\nu_k^{BP}, u_k^{x,BP} \}_{k=1}^{m} \), we get a dependent system of linear relations. Namely, there exist \( \{\nu_k^{BP} \}_{k=1}^{m} \) such that

\[ \sum_{k=1}^{m} \nu_k^{BP} \cdot u_k^{x,BP} = 0. \]

In other words, \( \sum_{k=1}^{m} \nu_k^{BP} \cdot r_k^{x,BP} \cdot r_{\mathcal{T}}^{BP} = 0 \mod p \), 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 \( \sum_{k=1}^{m} \nu_k^{BP} \cdot r_k^{x,BP} \cdot r_{\mathcal{T}}^{BP} \neq 0 \mod p \) where \( r_k^{x,BP} \) 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 \( \prod_{j=0}^{\ell-1} A_{j,x[\text{inp}(i)]} \) and \( \prod_{j=\ell+1}^{\ell} A_{j,x[\text{inp}(i)]} \) in \( s_x \). Two branching programs that differ from each other in this sense are referred to as **partially inequivalent**.

**What Programs are Partially Inequivalent? Attack on NC1 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 NC1 circuits. However, due to complex structures of such programs the analysis becomes quite non-trivial. 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 NC1 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

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

\[ \text{Note that, the analysis of Miles et al. uses } 2 \times 2 \text{ matrices in addition to using simple branching programs. These simplifications allow them to base their analysis on many facts related to the structures of these programs. Our aim here is to see if the attack works for programs obtained from NC1 circuits, in particular via Barrington’s Theorem. So, unfortunately it is not clear if their approach can be applicable here as the structure of the programs yielded via Barrington’s Theorem become much complex structurally (and also much larger in size) to analyze.} \]
all possible 16 inputs. Furthermore, we can escalate the attack to any pair of $\text{NC}^1$ circuits $(E_0, E_1)$ where $E_b = \neg C_b \land D_b$ ($b \in \{0, 1\}$) for practically any two $\text{NC}^1$ 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 $\neg C_b \land D_b$ can be effectively “collapsed” to a much smaller circuit $\neg C_b \land 0$ (0 refers to the smallest trivial circuit consisting of only identities). Finally, again via our Sage-implementation we show that for circuits $\neg C_0 \land 0$ and $\neg C_1 \land 0$ 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 $\text{NC}^1$ 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 $[n] \overset{\Delta}{=} \{1, \ldots, n\}$.

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 $A[i, \ast]$, its $j$-th column by $A[\ast, j]$ and the element in the $i$-th row and $j$-th column by $A[i, j]$. The $i$-th element of a vector $v$ is denoted by $v[i]$.

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 $A = (\text{inp}, A_0, \{A_{i, b}\}_{i \in [\ell], b \in \{0, 1\}}, A_{\ell + 1})$ be a single-input branching program of matrix-dimension $d$ and length $\ell$ over $n$-bit input.

---

14We 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 $\phi_{A,x}^{(i)}$ as follows:

$$
\phi_{A,x}^{(i)} \defeq \begin{cases}
(A_0 \cdot \prod_{j=1}^{i-1} A_{j,x[\inp(j)]} \cdot A_{\ell+1})^T \in \{0,1\}^{1 \times d^2} & \text{if } i \in [\ell], \\
\left(\prod_{j=1}^{\ell} A_{j,x[\inp(j)]} \cdot A_{\ell+1}\right)^T \in \{0,1\}^{1 \times d} & \text{if } i = 0,
\end{cases}
$$

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

$$
\gamma_{A,x}^{(i)} \defeq \begin{cases}
[\phi_{A,x}^{(i)} \mid 0^{d^2}] & \text{if } i \in [\ell] \text{ and } x[\inp(i)] = 0, \\
[0^{d^2} \mid \phi_{A,x}^{(i)}] & \text{if } i \in [\ell] \text{ and } x[\inp(i)] = 1, \\
\phi_{A,x}^{(i)} & \text{if } i = 0 \text{ or } \ell + 1,
\end{cases}
$$

where $\inp$ is a function from $[\ell] \to [n]$ and that $x[\inp(i)]$ denotes the bit of $x$ corresponding to location described by $\inp(x)$.

2. Then the linear partial product vector $\phi_{A,x}$ and the quadratic partial product vector $\psi_{A,x}$ of $A$ with respect to $x$ are defined as:

$$
\phi_{A,x} \defeq \left[\gamma_{A,x}^{(0)} \mid \cdots \mid \gamma_{A,x}^{(\ell+1)}\right] \in \{0,1\}^{1 \times (2d+2^d \cdot d^2)},
$$

$$
\psi_{A,x} \defeq \phi_{A,x} \otimes \phi_{A,x} \in \{0,1\}^{1 \times (2d+2^d \cdot d^2)^2},
$$

where $x = \overline{x} \oplus 1^n$ 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 $\Phi_{A,X}$ and the quadratic partial product matrix $\Psi_{A,X}$ of $A$ with respect to $X$ are defined as:

$$
\Phi_{A,X} \defeq \left[\begin{array}{c}
\phi_{A,x_1} \\
\phi_{A,x_2} \\
\vdots \\
\phi_{A,x_m}
\end{array}\right] \in \{0,1\}^{m \times (2d+2^d \cdot d^2)},
$$

$$
\Psi_{A,X} \defeq \Phi_{A,X} \otimes \Phi_{A,X} + \Psi_{A,X} \otimes \Psi_{A,X} = \left[\begin{array}{c}
\psi_{A,x_1,\overline{\pi}} + \psi_{A,\pi_1} \\
\psi_{A,x_2,\overline{\pi}} + \psi_{A,\pi_2} \\
\vdots \\
\psi_{A,x_m,\overline{\pi}} + \psi_{A,\pi_m}
\end{array}\right] \in \{0,1\}^{m \times (2d+2^d \cdot d^2)^2},
$$

where $\overline{X} \defeq \{\overline{x}_1, \overline{x}_2, \ldots\}$.

**Definition 2 (Partial Inequivalence).** Let $A_0$ and $A_1$ be two single-input matrix branching programs of matrix-dimension $d$ and length $\ell$ 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 $A_0(x) = A_1(x) = 0$ and $A_0(\overline{x}) = A_1(\overline{x}) = 0$.
- $\colsp(\Psi_{A_0,X}) \neq \colsp(\Psi_{A_1,X})$. 
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 GGHi3 setting in the full version.

\[ \text{Theorem 3.} \ Let O be the generic obfuscator described in Section 3.2 of the full version. Then for any two functionally equivalent same length single-input branching programs } A_0, A_1 \text{ that are partially inequivalent there exists a probabilistic polynomial time attacker that distinguishes between } O(A_0) \text{ and } O(A_1) \text{ with noticeable probability in the abstract attack model.} \]

\[ \text{Proof.} \]

\[ \text{Setup for the attack} \]

The given branching programs \( A_0 \) and \( A_1 \) are provided to be functionally equivalent and partially inequivalent. Therefore there exists a set \( X \) such that: (1) for all \( x \in X, A_0(x) = A_1(x) = 0 \), and (2) \( \text{colsp}(\Psi_{A_0,x}) \neq \text{colsp}(\Psi_{A_1,x}) \). We will assume that the adversary has access to \( X \) as auxiliary information.

\[ \text{Challenge} \]

\( A \) receives as a challenge the obfuscation of the branching program: \( A \in \{A_0, A_1\} \) by the challenger. Recall from the description of the abstract obfuscator that, the obfuscation of program \( A = (\text{inp}, A_0, \{A_{i,b}\}_{i \in [\ell], b \in \{0,1\}}, A_{\ell+1}) \), denoted by \( O(A) \) consists of the following public variables:

\[ Y_0 := A_0 \cdot R_1 \cdot 0 + gZ_0, \quad Y_{i,b} := \alpha_{i,b} R_i \cdot A_{i,b} \cdot R_{i+1} + gZ_{i,b}, \quad Y_0 := R_{\ell+1} \cdot A_{\ell+1} + gZ_0, \]

where the arbitrary secret variables are:

\[ \tilde{A}_0 \overset{\text{def}}{=} A_0 \cdot R_1 \overset{\text{adj}}{=} R_1 \cdot 0, \quad \tilde{A}_{i,b} \overset{\text{def}}{=} \alpha_{i,b} (R_i \cdot A_{i,b} \cdot R_{i+1}) \overset{\text{adj}}{=} R_{i+1} \cdot A_{\ell+1}; \]

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:

\[ Y_0 := (A_0 + gZ_0) \cdot R_1 \overset{\text{adj}}{=} R_1 \cdot 0, \quad Y_{i,b} := \alpha_{i,b} R_i \cdot (A_{i,b} + gZ_{i,b}) \cdot R_{i+1}, \quad Y_0 := R_{\ell+1} \cdot (A_{\ell+1} + gZ_{\ell+1}). \]

\[ \text{Pre-Zeroizing Computation (Type-1 queries)} \]

On receiving the obfuscation of \( A \in \{A_0, A_1\}, O(A) = \{Y_0, \{Y_{i,b}\}, Y_{\ell+1}\} \) the attacker, in the pre-zeroizing step, performs a “valid” Type-1 queries on all the inputs \( X, X \) where \( X = \{x_1, \ldots, x_m\}, X = \{x_1, \ldots, x_m\} \). That is, for an \( x \in \{0,1\}^n \), and the abstract obfuscation \( O(A) \), the attacker queries the polynomial:

\[ P_{A,x} = y_0 \cdot \prod_{i=1}^{\ell} Y_{i,x[\text{inp}(i)]} \cdot Y_{\ell+1}. \]

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

\[ P_{A,x} = P_{A,x}^{(0)}(\{Y_i\}_i) + g \cdot P_{A,x}^{(1)}(\{Y_i\}_i) + \ldots + g^{\ell+2} \cdot P_{A,x}^{(\ell+2)}(\{Y_i\}_i). \]
for some polynomials \( P^{(j)}_{A,x}(\{Y_i\}) \) \((j \in \{0, \ldots, \ell + 1\})\). However, by Lemma 4 we have that:

\[
P^{(0)}_{A,x} = \rho \tilde{\alpha}_x A(x)
\]

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

\[
W_{A,x} = P^{(1)}_{A,x} / g = P^{(1)}_{A,x} + g \cdot P^{(2)}_{A,x} + \ldots + g^{\ell+1} \cdot P^{(\ell+2)}_{A,x}.
\]

Analogously, the attacker obtains handles \( W_{A,\pi} \). After obtaining handles

\[
\{(W_{A,x_1}, W_{A,\pi_1}), \ldots (W_{A,x_m}, W_{A,\pi_m})\}
\]

the attacker starts the post-zeroizing phase.

**Post-Zeroizing Computation**

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

(i) \( Q^{\text{ann}}(P^{(1)}_{A_{b,x_1}, \pi_1}, \ldots, P^{(1)}_{A_{b,x_m}, \pi_m}) \overset{\text{def}}{=} 0. \)

(ii) \( Q^{\text{ann}}(P^{(1)}_{A_{1-x_1}, \pi_1}, \ldots, P^{(1)}_{A_{1-x_m}, \pi_m}) \not\overset{\text{def}}{=} 0. \)

Clearly, this leads to an attack on the obfuscation security as \( A \) would receive 0 from the challenger if and only if \( Q^{\text{ann}}(P^{(1)}_{A_{b,x_1}, \pi_1}, \ldots, P^{(1)}_{A_{b,x_m}, \pi_m}) \) is identically zero, hence it would receive 0 if and only if \( A_b \) is chosen by the challenger in the challenge phase. To find such \( Q^{\text{ann}} \) the attacker continues as follows. Observe that by Lemma 4, for every \( x \in X \) we have that:

\[
P^{(1)}_{A,x} = \rho \tilde{\alpha}_x (\phi_{A,x} \cdot z^T)',
\]

\[
P^{(1)}_{A,\pi} = \rho \tilde{\alpha}_\pi (\phi_{A,\pi} \cdot z^T).
\]

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

\[
\tilde{P}^{(1)}_{A,x} \overset{\text{def}}{=} P^{(1)}_{A,x} P^{(1)}_{A,\pi} = \rho^2 \tilde{\alpha} ((\phi_{A,x} \cdot z) \otimes (\phi_{A,\pi} \cdot z))
\]

\[
= \rho^2 \tilde{\alpha} ((\phi_{A,x} \otimes \phi_{A,\pi}) \cdot (z^T \otimes z^T))
\]

\[
= \rho^2 \tilde{\alpha} (\psi_{A,x} \cdot z^T \otimes z^T).
\]

where \( \tilde{\alpha} \overset{\text{def}}{=} \tilde{\alpha}_x \tilde{\alpha}_\pi \) is now independent of input \( x \).\(^{15}\) Similarly we can also have:

\[
\tilde{P}^{(1)}_{A,\pi} \overset{\text{def}}{=} P^{(1)}_{A,\pi} P^{(1)}_{A,x} = \rho^2 \tilde{\alpha} ((\phi_{A,\pi} \cdot z) \otimes (\phi_{A,x} \cdot z))
\]

\[
= \rho^2 \tilde{\alpha} ((\phi_{A,\pi} \otimes \phi_{A,x}) \cdot (z^T \otimes z^T))
\]

\[
= \rho^2 \tilde{\alpha} (\psi_{A,\pi} \cdot z^T \otimes z^T).
\]

---

\(^{15}\) Here, we use the fact that the branching programs are single-input. For multi-input programs we do not know how to make \( \tilde{\alpha} \) 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:

\[ \tilde{P}_{A,x}^{(1)} + F_{A,x}^{(1)} = 2F_{A,x}P_{A,x}^{(1)} = \rho^2\hat{\alpha}(\psi_{A,x} \cdot z^T \otimes z^T) + \rho^2\hat{\alpha}(\psi_{A,x} \cdot z^T \otimes z^T) = \rho^2\hat{\alpha}(\psi_{A,x} + \psi_{A,x}) \cdot (z^T \otimes z^T). \]

Using the given conditions that \( \Psi_{A_i,X} \) and \( \Psi_{A_j,X} \) have distinct column spaces (and hence distinct left-kernel) the attacker can efficiently compute (e.g. via Gaussian Elimination) a vector \( v_{\text{ann}} \in \{0,1\}^{1 \times m} \) that belongs to its left-kernel, call it the annihilating vector, such that for some \( b \in \{0,1\} \) we have:

\[ v_{\text{ann}} \cdot \Psi_{A_i,X} = 0 \quad \text{but} \quad v_{\text{ann}} \cdot \Psi_{A_{1-b},X} \neq 0. \]

The corresponding annihilation polynomial \( Q_{\text{ann}} \) can be written as:

\[ Q_{\text{ann}}(W_{A,x}, W_{A,\tau_1}, \ldots, W_{A,x_m}, W_{A,\tau_m}) = v_{\text{ann}} \cdot \begin{bmatrix} W_{A,x}, W_{A,\tau_1} \\ \vdots \\ W_{A,x_m}, W_{A,\tau_m} \end{bmatrix}. \]

Observe that the coefficient of \( g^0 \) in the expression \( Q_{\text{ann}}(W_{A,x}, W_{A,\tau_1}, \ldots, W_{A,x_m}, W_{A,\tau_m}) \) from above is equal to \( Q_{\text{ann}}(P_{A_i,x}, P_{A_i,\tau_1}, \ldots, P_{A_i,x_m}, P_{A_i,\tau_m}) \). Moreover this value for \( A = A_b \) is:

\[ Q_{\text{ann}}(P_{A_i,x}, P_{A_i,\tau_1}, \ldots, P_{A_i,x_m}, P_{A_i,\tau_m}) = v_{\text{ann}} \cdot \frac{\Psi_{A_b,X} - z^T}{2} \cdot (z \otimes z)^T \equiv 0 \]

but for \( A_{1-b} \):

\[ Q_{\text{ann}}(P_{A_i,x}, P_{A_i,\tau_1}, \ldots, P_{A_i,x_m}, P_{A_i,\tau_m}) = v_{\text{ann}} \cdot \frac{\Psi_{A_b,X} - z^T}{2} \cdot (z \otimes z)^T \neq 0. \]

Hence, the response to Type 2 query is sufficient to distinguish between obfuscation of \( A_b \) and \( A_{1-b} \) in the abstract model. This concludes the proof.

\section*{Evaluations of \( P_{A,x}^{(0)} \) and \( P_{A,x}^{(1)} \)}

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

\begin{lemma}
For every \( x \in \{0,1\}^n \), we have that:
\[ P_{A,x}^{(0)} = \rho\tilde{\alpha}_x A(x), \]
\[ P_{A,x}^{(1)} = \rho\tilde{\alpha}_x (\phi_{A,x} \cdot z)^T, \]
\end{lemma}

where \( \rho \) is defined as \( \prod_i \det(R_i) \) and \( \tilde{\alpha}_x \) is defined as \( \prod_{i=1}^t \alpha_{i-x_{\text{eq}(i)}} \) and \( z \) is a vector consisting of the random terms \( Z_0, Z_{1-b}, \) and \( Z_{1+b} \) used to generate the obfuscation terms \( Y_0, Y_{1-b}, \) and \( Y_{1+b} \) in an appropriate sequence.

\section*{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 $\mathsf{NC}^1$ circuits such that the corresponding Barrington-implemented\(^\text{16}\) 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 $\{0,1\}^4 \rightarrow \{0,1\}$:

- $C_0(x) \overset{\text{def}}{=} (x[1] \land 1) \land (x[2] \land 0) \land (x[3] \land 1) \land (x[4] \land 0)$,
- $C_1(x) \overset{\text{def}}{=} (x[1] \land 0) \land (x[2] \land 0) \land (x[3] \land 0) \land (x[4] \land 0)$.

Define the set $X \overset{\text{def}}{=} \{0,1\}^4$. Now, we provide an implementation (see the full version \cite{4} for more details on the implementation) in Sage \cite{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:

\begin{itemize}
  \item **Claim 5.** Let $A_{C_0}, A_{C_1}$ be the Barrington-Implementation of the circuits $C_0, C_1$ respectively, then we have that: $\text{colsp} (\Psi_{A_{C_0},X}) \neq \text{colsp} (\Psi_{A_{C_1},X})$.
\end{itemize}

\begin{itemize}
  \item **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 \cite{4} for the details of our implementation. Throughout this section we refer to this particular Barrington-implementation.
\end{itemize}

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

- $E_0(y) \overset{\text{def}}{=} \neg C_0(x) \land D_0(x')$,
- $E_1(y) \overset{\text{def}}{=} \neg C_1(x) \land D_1(x')$,

($\neg C$ 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 $Y \overset{\text{def}}{=} \{x \circ x^* \mid x \in \{0,1\}^4\}$ (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 $A_{E_0}, A_{E_1}$ be the Barrington-implementations of $E_0, E_1$ respectively, then we have that: $\text{colsp}(\Psi_{A_{E_0},Y}) \neq \text{colsp}(\Psi_{A_{E_1},Y})$.

6.3 Partially Inequivalent Universal Circuits

In this section we present constructions of $(\text{NC}^1)$ universal circuits that, when compiled with two arbitrary distinct $(\text{NC}^1)$ 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 $\text{NC}^1$ universal circuits $U = \{U_1, U_2, \ldots, U_v\}$ of size $v = O(\text{poly}(\lambda))$ such that: given two arbitrary functionally equivalent $\text{NC}^1$ circuits $G_0, G_1$ that compute arbitrary boolean function $\{0,1\}^n \rightarrow \{0,1\}$ satisfying (i) $|G_0| = |G_1| = v$ and (ii) there exists an input $x^*$ such that $G_0(x^*) = G_1(x^*) = G_0(x^*) = G_1(x^*) = 0$; 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


38:14 Cryptanalysis of Indistinguishability Obfuscations of Circuits over GGH13


