More Annihilation Attacks

an extension of MSZ16

Charles Jin

Yale University

May 9, 2016

1 Introduction

MSZ16 gives an annihilation attack on candidate constructions of indistinguishability obfuscation over GGH13, but only for so-called "trivial" branching programs. Here we extend that attack to a larger class of programs.

1.1 Overview

The annihilation attack is unique in that it doesn't not require the scheme to reveal any low-level encodings of zeros, which previous attacks have depended on. Rather, the attack uses the public zero-testing procedure to generate many top-level encodings of zeros. Then, it uses a special annihilating polynomial which is identically zero when evaluated over the top-level encodings of zero for one distribution, but non-zero for a different distribution. In such a way, it is able to distinguish between encodings of two different sets of plaintext elements. Put another way, the encoding scheme leaks information about its underlying plaintext elements via the zero-testing procedure.

Attack setting. The attacker is given a vector $\vec{u}$ of l encodings, as well as two vectors $\vec{\alpha}^{(0)}$ and $\vec{\alpha}^{(1)}$, which are each seen as distributions of l values to be encoded, each with respect to a vector $\vec{r}$ of l random small elements. The attack is then to determine which bit $b \in \{0, 1\}$ corresponds to the underlying $\vec{\alpha}^{(b)}$ of the encoded values $\vec{u}$.

Finally, instead of viewing the $\vec{\alpha}^{(b)}$ as distributions over values, we view them as vectors of multivariate polynomials over some underlying variables that yield the distributions. For the remainder of this section we will consider the $\vec{\alpha}^{(b)}$ as vectors of polynomials.

Brief description. Recall that we are not given any access to any low-level encodings of zero, so the only thing we can do is algebraic manipulations of lower-level encodings to build a top-level encoding of zero, then multiply by the zero-testing parameter.

Assume that we have such an encoding of zero e. Recall that an element is of the form

$$u_i = \left[\frac{\alpha_i + gr_i}{z^{j_i}}\right], i \in [l].$$

Then by definition e is in the ideal generated by g, so it is of the form:

$$e = \frac{g\gamma_1 + g^2\gamma_2 + ... + g^k\gamma_k}{z^k}.$$

The results of multiplying by the zero-testing parameter thus yields

$$f := [e \cdot p_{zt}]_q = h \cdot (\gamma_1 + g\gamma_2 + ... + g^{k-1}\gamma_k),$$

where each γ_i is a polynomial in entries from $\vec{\alpha}$ and $\vec{r}$.

Our distinguishing attack will be to develop a small polynomial that "annihilates" several of these. In particular, in our ring $R/\mathcal{I}$, f is contained in the coset $h \cdot \sigma_i$. If we require our annihilating polynomial to be homogenous (which is always possible, as will be explained later), then we merely need some polynomial that is identically zero on some set of f.

Annihilating polynomials—an example. Say that we have three top-level encodings of zero e', e'', e''' and multiplying by the zero-testing parameters yields f', f'', f'''. Assume further that for $\vec{\alpha}^{(0)}$ these correspond to $\gamma_1' = xr$, $\gamma_1'' = xr^2$, and $\gamma_1''' = x$, where x is some random variable underlying $\vec{\alpha}^{(0)}$. It is clear that $Q(a, b, c) := a^2 - bc$ *annihilates* $\gamma_1', \gamma_1'', \gamma_1'''$ in the sense that $Q(\gamma_1', \gamma_1'', \gamma_1''') = 0$. However, since a different polynomial underlies $\vec{\alpha}^{(1)}$, one could imagine e', e'', e''' giving rise to a different set of $\gamma_1', \gamma_1'', \gamma_1'''$, say x^3r, xr, x. Then $Q(\gamma_1', \gamma_1'', \gamma_1''') = x^6r^2 - x^2r$.

The distinguishing attack thus tests $Q(f', f'', f''') \stackrel{?}{\in} \langle hg \rangle$. Because the polynomial is homogenous, if $Q(\gamma_1', \gamma_1'', \gamma_1''')$ is the zero polynomial, then so is $Q(f', f'', f''')$. Thus, if the challenge bit is 0, then $Q(f', f'', f''')$ is in the ideal generated by $\langle hg \rangle$; otherwise (if the challenge bit is 1) it isn't.

1.2 Model description

First we develop a toy version of an abstract model for attacks; later, we will see that this easily generalizes to the actual setting of GGH13. First, we pick some field $\mathbb{F}$ over which our variables will be defined. Then, for some integers n and m, there are "hidden" variables $X_1, ...X_n$, $Z_1, ...Z_n$, and g. The "public" variables $Y_1, ..., Y_m$ are then set to $Y_i = q_i(\{X_j\}) + gZ_i$ for some polynomials q_i.

The adversary is allowed to make two kinds of queries:

1. In a so-called **Type 1** query, the adversary is able to submit some "valid" polynomial p_k on the public variables Y_i. Here, a "valid" polynomial is a restricted set of polynomials that result in valid calculations in the graded encoding system.

   In order to evaluate this query, consider p_k as a polynomial in g with formal variables X_j, Z_i, and g. We write $p_k = p_k^{(0)}(\{X_j\}, \{Z_i\}) + g p_k^{(1)}(\{X_j\}, \{Z_i\}) + g^2 ....$ If p_k is not identically 0 and $p_k^{(0)}$ is identically zero, then the adversary receives a handle to the new variable $W_k = p_k/g = p_k^{(1)}(\{X_j\}, \{Z_i\}) + g p_k^{(2)}(\{X_j\}, \{Z_i\}) + g^2 ....$ Otherwise, the adversary learns nothing.

   Note that for this exercise to be interesting, both distributions must return zero (and non-zero) for all efficiently-computable top-level encodings, otherwise the adversary could just distinguish by zero-testing.

2. In a **Type 2** query, the adversary can submit arbitrary polynomials r on the W_k seen thus far. Again we consider $r(W_k)$ as a polynomial in g with formal variables X_j, Z_i, and g, so we can write $r = r^{(0)}(\{X_j\}, \{Z_i\}) + g r^{(1)}(\{X_j\}, \{Z_i\}) + g^2 ....$ If $r^{(0)}$ is identically 0, then the response is 0; otherwise, the response is 1.

1.3 Obfuscation

In this section, I will describe the abstract obfuscation scheme that will be attacked. We take as input a branching program with parameters length l, input length n, and arity d. In other words, The branching program has l levels, and at each level, the program examines d bits of the n-bit long input to determine the next "step". This behavior is specified by an input function $inp : [l] \to 2^{[n]}$ with $|inp(i)| = d \ \forall i \in [l]$. A "step" is modeled by matrix multiplication, which each step selecting one of 2^d matrices to append to the product. Another way to think about this is to associated each input x with a set $T \subseteq [n]$, where $i \in T \iff x_i = 1$. Then if we write the matrices $\{A_{i,S_i}\}_{i \in [l]}$, with S_i ranging over the subsets of $inp(i)$, we say that we select a matrix $A_{i,Z}$ at step i if $Z = T \cap inp(i)$.

There are also two special "bookend" matrices A_0 and A_{l+1}. So evaluating the branching program on input x or set T is to compute the following product:

$$A(T) = A_0 \times \prod_{i=1}^{l} A_{i, T \cap inp(i)} \times A_{l+1}$$

The output of the branching program is thus 0 if and only if this product is 0.

In order to obfuscate this branching program, we generate $l + 2$ random matrices $\{R_i\}_{i \in [l+1]}$, one for each level plus the two bookend vectors, as well as one random scalar $\{\alpha_{i,S_i}\}_{i \in [l], S_i \subseteq inp(i)}$ for the $2^d l$ matrices at every level corresponding to each choice. The randomized branching program thus consists of matrices $\widetilde{A_{i,S_i}} = \alpha_{i,S_i}(R_i \cdot A_{i,S_i} \cdot R_{i+1}^{adj}$ and bookend matrices $\widetilde{A_0} = A_0 \cdot R_i^{adj}$ and $\widetilde{A_{l+1}} = R_{l+1} \cdot A_{l+1}$. It should be clear that the randomized branching program $\widetilde{A}(T)$ is zero if and only if $A(T) = 0$.

Finally, as in the model, the "hidden" variables are the $\{\widetilde{A}\}$ and the aforementioned g and $\{Z\}$. The "public" variables are then set to $Y_{i,S} = \widetilde{A_{i,S}} + gZ_{i,S}$. A **Type 1** is just evaluating the branching program on some input, which must be allowed.

2 An illustrative attack on "trivial" branching programs

To demonstrate the attack, we begin by attacking a trivial program.

2.1 The two branching programs.

We define two branching programs as follows. Each program has $2n + 2$ layers. The bookend matrices are the vectors $A_0 := \begin{pmatrix} 0 & 1 \end{pmatrix}$ and $A_{2n+1} := \begin{pmatrix} 1 & 0 \end{pmatrix}^T$. The middle $2n$ layers scan each input bit twice, once going forward then once going backward, so that the input selection function $inp(i) := \min(i, 2n + 1 - i)$.

For the trivial program A, both matrices at every layer are the 2-by-2 identity. For the second branching program A', the matrix at every layer reading input bit $i \leq k$ for some $k \leq n$ when the input bit is 0 is the 2-by-2 anti-diagonal matrix, while the case where the input bit is 1 is again the 2-by-2 identity matrix. In other words,

$$A'_{i,0} = A'_{2n+1-i,0} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \text{ for } i \in [k], k \leq n.$$

Clearly for both branching programs, $A(x) = A'(x) = 0$ for all x.

2.2 The attack.

We have now set up two different distributions of underlying inputs. A input x corresponds to hidden encoded bottom-level elements; evaluating the branching program on an input evaluates the multilinear map on these bottom-level elements and in all cases outputs a top-level encoding of zero. A distinguishing attack thus is able to determine the branching program, corresponding to the underlying distribution, by examining the (theoretically equivalent) outputs.

The attack proceeds in two phases:

1. In the first phase, the adversary submits **Type 1** queries and gets back handles (or nothing). In our case, a **Type 1** query is merely a run of the branching program.

2. In the second phase, the adversary submits a **Type 2** query in the form of a small annihilating polynomial for evaluation over the results returned by the **Type 1** queries. The annihilating polynomial is so-called because it is identically zero when evaluated over the results of the **Type 1** queries of one distribution, but non-zero when evaluated over the results of the other distribution.

The bulk of the attack thus rests on proving the existence, and the efficient construction, of such an annihilating polynomial. We begin by examining the form of the response to **Type 1** queries. Because both branching programs always return 0, every **Type 1** query should return a handle.

First, we adapt our model to the single-input case, instead of ranging the matrices A over sets S, we can take it over a single bit b. We first randomize each A by picking random invertible 2-by-2 matrices $\{R_i\}_{i\in[2n+1]}$ and non-zero scalars $\{\alpha_{i,x}\}_{i\in[2n],b\in\{0,1\}}$. As before, we randomize by setting $\widetilde{A_0} = A_0 \cdot R_1^{adj}$, $\widetilde{A_{2n+2}} = R_{2n_1} \cdot A_{2n+2}$, and $\widetilde{A_{i,b}}$, and finally, we encode each entry via GGH13 as follows:

$$Y_{i,b} = \alpha_{i,b} R_i \cdot A_{i,b} \cdot R_{i+1}^{adj} + gZ_{i,b}.$$

Next, because all the α, R are invertible, we can actually absorb these into the Z to write

$$Y_{i,b} = \alpha_{i,b} R_i \cdot (A_{i,b} + gZ'_{i,b}) \cdot R_{i+1}^{adj}.$$

Our **Type 1** queries will take an input x and be of the form

$$p_x = Y_0 \times \prod_{i=1}^{2n} Y_{i,T\cap inp(i)} \times Y_{2n+1}.$$

Because our branching programs always return 0, we know that the coefficient of g^0 in p_x is always zero. Let ρ satisfies $\rho I = \prod_i R_i R_i^{adj}$. Now we can get $p_x^{(1)}$, the coefficient of g^1, by collecting all the $\alpha_{i,x_{inp(i)}}$ and ρ at the front as a constant multiplicative factor. Then, for each i, we get a single $gZ_{i,x_{inp(i)}}$ from the product, and use the $A_j, x_{inp(j)}$ for all $j \neq i$. The zero-testing procedure reduces this by a factor of g, so on return, we get a handle to the variable:

$$p_x = \rho\Big(\prod_i \alpha_{i,x_{inp(i)}}\Big) \sum_{i=1}^{2n} \Big(\cdots A_{i-1,x_{inp(i-1)}} \cdot Z_{i,x_{inp(i)}} \cdot A_{i+1,x_{inp(i+1)}} \cdots \Big) + g\Big(\cdots\Big),$$

The rest of the proof will to be to find a small annihilating polynomial on some set of $p_x^{(1)}$ for various inputs x. If this quantity is identically zero, then the annihilating polynomial will also return zero on the full handles, since everything left has a factor of g. Thus we drop the rest of the terms with a g in them and just consider $p_x^{(1)}$.

Recall that we can associate an input $x \in \{0,1\}^n$ with a set $T \subseteq [n]$ where $i \in T \iff x_i = 1$. Since we know the branching programs scan the same input bits both forward and backward, we can pair those coefficients together. Thus, let $\alpha'_{i,b} = \alpha_{i,b}\alpha_{2n+1-i,b}$ for $i \in [2, n]$ and let $\alpha'_{1,b} = \rho\alpha_{1,b}\alpha_{2n,b}$. and

$$U_x = U_T = \textstyle\prod_i \alpha'_{i,x_i} = \rho \prod_i \alpha_{i,x_{inp(i)}}$$

The following is stated without proof; a more general version is proved later.

Claim 2.1. *For $|T| \geq 2$, the following identity holds:*

$$U_T = \prod_{S \subseteq T, |S| \leq 1} U_S^{\left[\binom{|T|-|S|-1}{1-|S|}(-1)^{|S|+1}\right]}$$

Now we consider the other half of the $p_x^{(1)} = p_T^{(1)}$. We have two cases:

1. If the branching program is all-identity, i.e. the trivial branching program, then we have $A_{i,0} = A_{i,1} = A_i$. Define $\beta_{i,b} = \cdots A_{i-1} \cdot Z_{i,x_{inp(i)}} \cdot A_{i+1} \cdots$. As with before, we know that we read all the input bits twice, so write $\beta'_{i,b} = \beta_{i,b} + \beta_{2n+1-i,b}$. Then we can define

$$V_x = V_T = \sum_{i=1}^{n} \beta'_{i,x_i} = \sum_{i=1}^{2n} \Big(\cdots A_{i-1,x_{inp(i-1)}} \cdot Z_{i,x_{inp(i)}} \cdot A_{i+1,x_{inp(i+1)}} \cdots \Big).$$

   Clearly then $p_T^{(1)} = U_T V_T$.

2. If the branching program has the anti-diagonal matrices for $b = 0, i \leq k$, then we consider the term $\big(\cdots A_{i-1,x_{inp(i-1)}} \cdot Z_{i,x_{inp(i)}} \cdot A_{i+1,x_{inp(i+1)}} \cdots \big)$ in three cases.

   (a) First, assume that $i \leq k+1$. Then since all the $A_{i,b}$ are either identity or anti-diagonal, depending on the parity of $x_{inp(i)}$, the product up to $\big(\cdots A_{i-1,x_{inp(i-1)}} \big) = \big(\cdots A_{i-1,x_{i-1}} \big)$ depends on the parity of $x_{[1,i-1]} := \prod_{j=1}^{i-1} x_j$. In particular, if the parity of $x_{[1,i-1]}$ is zero, then the product is just the row vector $\begin{pmatrix} 0 & 1 \end{pmatrix}$, and if it is 1, then the product is $\begin{pmatrix} 1 & 0 \end{pmatrix}$. Similarly, the product $\big(A_{i+1,x_{inp(i+1)}} \cdots \big) = \big(A_{i+1,x_{i+1}} \cdots \big)$ is $\begin{pmatrix} 1 & 0 \end{pmatrix}^T$ if the parity is 0, and $\begin{pmatrix} 0 & 1 \end{pmatrix}^T$ otherwise. Thus, the whole product is equal to

$$\begin{cases} \left(Z_{i,x_{inp(i)}}\right)_{1,2} = \begin{pmatrix} 0 & 1 \end{pmatrix} \left(Z_{i,x_{inp(i)}}\right) \begin{pmatrix} 1 & 0 \end{pmatrix}^T & \text{if the parity is 0} \\ \left(Z_{i,x_{inp(i)}}\right)_{2,1} = \begin{pmatrix} 1 & 0 \end{pmatrix} \left(Z_{i,x_{inp(i)}}\right) \begin{pmatrix} 0 & 1 \end{pmatrix}^T & \text{if the parity is 1} \end{cases}$$

   Thus we define $\gamma_{i,b,p}$ to be the product when $x_i = x_{inp(i)} = b$ and the parity of $x_[1, i-1]$ is p.

   (b) Next, if $i \in [2n+k, 2n]$, we get the same thing, so that i is absorbed into the same $\gamma_{i,b,p}$ as the previous case.

   (c) Finally, if $i \in [k+1, 2n-k-1]$, it is almost the same thing, except we only care about the parity of bits $x_{[1,k]}$ (since the rest of the matrices are all identity and so the parity of the bits don't matter). Thus we can write $\gamma_{i,b,p}$ where $x_i = b$ as before, and the parity of $x_{[1,k]}$ is p.

Finally, define

$$W_x = W_T = \sum_{i=1}^{n} \gamma_{i,x_i,parity(x_{[1,min(i-1,k)]})}$$

We have $p_T^{(1)} = U_T W_T$.

Now, we can begin to build the annihilating polynomial. Notice that the number of W_T is 2^n, corresponding to each subset $T \subseteq [n]$, but the number of γ is only $4n$: n choices for the bit i, 2 choices for the parity of x_i, and 2 choices for the parity of b. Thus, there must be some linear relationship between the W_T. Further, if we fix the bits $x_1, ..., x_k$ to 0, then the parity of these bits is also fixed (to 0), so the W_T and V_T should satisfy the same equations for this T.

On the other hand, if $T \subseteq \{1, 2, 3\}$ and $k = 1$, the only linear relationships among the W_T are:

$$W_{1,2,3} + W_1 = W_{1,2} + W_{1,3}$$
$$W_{2,3} + W_\emptyset = W_2 + W_3$$

This is different and less than the equations satisfied by the V_T. Thus there exists some polynomial $Q_{1,2,3}$ on the set of $p_T^{(0)}$ for $T \subseteq \{1, 2, 3\}$ that annihilates in the all-identity case, but does not in the anti-diagonal case. See MSZ for a precise definition of such a polynomial.

The attack case be summarized thusly. First, evaluate the branching program on inputs x that are zero everywhere but the first 3 bits, and vary those over all subsets of [3]. This gives handles to elements $p_T^{(0)}$, with $T \subseteq \{1, 2, 3\}$. Then we evaluate the annihilating polynomial $Q_{1,2,3}$ on these handles. If the result is 0, then it w.h.p. it is the all-identity case; otherwise, we are in the anti-diagonal case.

3 An attack on larger branching programs

In the previous section, we considered two branching programs, one consisting of all identity matrices, and the other with the first and last layers examining the first bit and outputting either the identity or anti-identity matrix. In this section, we will move on to a pair of slightly more complicated branching programs.

3.1 The two branching programs.

The branching programs are defined similarly as in the first example, except that the matrices are 3-by-3. The programs each have $3n + 2$ layers, with bookend vectors $A_0 := \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}$ and $A_{3n+2} := \begin{pmatrix} 1 & 0 & 0 \end{pmatrix}^T$. The middle $3n$ layers scan each input bit three times,

once going forward, once going backward, and one more time forwards, so that the input selection function is

$$inp(i) := \begin{cases} i & 0 \le i \le n \\ 2n+1-i & n+1 \le i \le 2n \\ i-2n & 2n+1 \le i \le 3n \end{cases}$$

For the trivial program A, both matrices at every layer are the 3-by-3 identity. For the second branching program A', the matrix at every layer reading input bit $i \le k$ for some $k \le n$ when the input bit is 0 is a 3-by-3 permutation matrix, while the case where the input bit is 1 is again the 3-by-3 identity matrix. In other words,

$$A'_{i,0} = A'_{2n+1-i,0} = A'_{i+2n} \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \text{ for } i \in [k], k \le n.$$

Clearly for both branching programs, $A(x) = A'(x) = 0$ for all x. We further restrict the analysis to $k = 1$.

3.2 The attack.

As before, all we need is to find some small annihilating polynomial on some set of $p_x^{(1)}$ for various inputs x.

Recall that on a **Type 1** query we get a handle to the following variable:

$$p_x^{(1)} = \rho\Big(\prod_i \alpha_{i,x_{inp(i)}}\Big) \sum_{i=1}^{2n} \Big(\cdots A_{i-1,x_{inp(i-1)}} \cdot Z_{i,x_{inp(i)}} \cdot A_{i+1,x_{inp(i+1)}} \cdots \Big),$$

As before, we associate an input $x \in \{0,1\}^n$ with a set $T \subseteq [n]$ where $i \in T \iff x_i = 1$ (we denote the empty set 0). Since we know the branching programs scan the same input bits every time, we can group those coefficients together. Thus, let $\alpha'_{i,b} = \alpha_{i,b}\alpha_{2n+1-i,b}\alpha_{2n+i}$ for $i \in [2,n]$ and let $\alpha'_{1,b} = \rho\alpha_{1,b}\alpha_{2n,b}\alpha_{2n+1}$. Then we can define

$$U_x = U_T = \textstyle\prod_i \alpha'_{i,x_i} = \rho \prod_i \alpha_{i,x_{inp(i)}}$$

Claim 3.1. *For $|T| \ge 2$, the following identity holds:*

$$U_T = \prod_{S \subseteq T, |S| \le 1} U_S^{\left[\binom{|T|-|S|-1}{1-|S|}(-1)^{|S|+1}\right]}$$

Proof. Consider some such set $T = \{t_1, t_2, ...\}$. For $i = t_j \in T$ (i.e. $x_i = 1$), let us rewrite $\alpha'_{i,1} = x_{t_j}$ and $\alpha'_{i,0} = x'_{t_j}$, and for $i = t_j \notin T$, rewrite $\alpha'_{i,0} = y_{t_j}$. Then we have

$$U_T = \prod_i \alpha'_{i,x_i} = \prod_{t_j \in T} x_{t_j} \prod_{t_j \notin T} y_{t_j}.$$

Conversely, for $S \subseteq T$, we get

$$U_S = \prod_{t_j \in S \cap T} x_{t_j} \prod_{t_j \in T - S} x'_{t_j} \prod_{t_j \notin T} y_{t_j}.$$

Consider the three types of terms separately. For $S \subseteq T$, $|S| = 1$, we clearly cover the $\{x_{t_j}\}$ exactly once, i.e.

$$\prod_{t_j \in T} x_{t_j} = \prod_{S \subseteq T, |S|=1} \prod_{t_j \in S} x_{t_j}.$$

However, for $|S| = 1$, we have over-counted the other y_{t_j} terms, as well as introduced some x'_{t_j} terms. First, consider the y_{t_j} terms. Notice that every U_S has exactly one copy. For $|S| = 1$, there are exactly T such sets, and the exponent on these terms is 1, thus accumulating T copies. Since we need only need 1 copy, this leaves $T - 1$ extra copies of the y_{t_j}. Additionally, every x'_{t_j} term shows up for every $S, |S| = 1$ except for the one for the corresponding x_{t_j} (i.e. the term U_S where $S = \{x_{t_j}\}$. This leaves, again, $T - 1$ extra copies of the x'_{t_j}. Putting this all together, we get

$$\begin{aligned}
\prod_{S \subseteq T, |S|=1} U_S &= \prod_{S \subseteq T, |S|=1} \Big(\prod_{t_j \in S \cap T} x_{t_j} \prod_{t_j \in T - S} x'_{t_j} \prod_{t_j \notin T} y_{t_j} \Big) \\
&= \prod_{t_j \in T} x_{t_j} \Big(\prod_{t_j \in T} x'_{t_j} \Big)^{|T|-1} \Big(\prod_{t_j \notin T} y_{t_j} \Big)^{|T|}
\end{aligned}$$

For $|S| = 0$, the set $t_j \in S$ is empty, so that

$$\begin{aligned}
U_S &= \prod \Big(\prod_{t_j \in S \cap T} x_{t_j} \prod_{t_j \in T - S} x'_{t_j} \prod_{t_j \notin T} y_{t_j} \Big) \\
&= \prod_{t_j \in T} x'_{t_j} \prod_{t_j \notin T} y_{t_j}
\end{aligned}$$

Thus we see that all we need to do is multiply together all the $U_S, |S| = 1$ and then divide by $|T| - 1$ copies of $U_S, |S| = 0$:

$$
\begin{aligned}
U_T &= \prod_{t_j \in T} x_{t_j} \prod_{t_j \notin T} y_{t_j} \\
&= \frac{\prod_{t_j \in T} x_{t_j} \Big(\prod_{t_j \in T} x'_{t_j} \Big)^{|T|-1} \Big(\prod_{t_j \notin T} y_{t_j} \Big)^{|T|}}{\Big(\prod_{t_j \in T} x'_{t_j} \Big)^{|T|-1} \Big(\prod_{t_j \notin T} y_{t_j} \Big)^{|T|-1}} \\
&= \Big(\prod_{S \subseteq T, |S|=1} U_S \Big) \Big(\prod_{S \subseteq T, |S|=0} U_S \Big)^{|T|-1} \\
&= \prod_{S \subseteq T, |S| \leq 1} U_S^{\left[\binom{|T|-|S|-1}{1-|S|} (-1)^{|S|+1} \right]}
\end{aligned}
$$

□

It is important to note that this is the same identity we get from the 2-by-2 case. This is because this part of the analysis is the same regardless of the matrices and comes only from the extra randomization terms $\alpha_{i,b}$.

Now we consider the other half of the $p_x^{(1)} = p_T^{(1)}$. We have two cases:

1. When the branching program is all-identity, we have $A_{i,0} = A_{i,1} = A_i$. Define $\beta_{i,b} = \cdots A_{i-1} \cdot Z_{i,x_{inp(i)}} \cdot A_{i+1} \cdots$. As with the analysis above, every input bit is read three times, so write $\beta'_{i,b} = \beta_{i,b} + \beta_{2n+1-i,b} + \beta_{2n+i,b}$. Then we can define

$$
V_x = V_T = \sum_{i=1}^{n} \beta'_{i,x_i} = \sum_{i=1}^{3n} \Big(\cdots A_{i-1,x_{inp(i-1)}} \cdot Z_{i,x_{inp(i)}} \cdot A_{i+1,x_{inp(i+1)}} \cdots \Big).
$$

   As with the U_T, we get a similar identity:

   Claim 3.2. *For $|T| \geq 2$, the following identity holds:*

$$
V_T = \sum_{S \subseteq T, |S| \leq 1} \Big[\tbinom{|T|-|S|-1}{1-|S|} (-1)^{|S|+1} \Big] V_S
$$

   The proof is analogous as the U_T, and is omitted. We have $p_T^{(1)} = U_T V_T$.

2. If the branching program has the permutation matrices for $b = 0$, $i = 1$, then denote the permutation matrix P_3. We consider the term $\big(\cdots A_{i-1,x_{inp(i-1)}} \cdot Z_{i,x_{inp(i)}} \cdot A_{i+1,x_{inp(i+1)}} \cdots \big)$ in six cases.

(a) $x_1 = 1$. Then all the matrices are identities and we get $\begin{pmatrix}0 & 0 & 1\end{pmatrix}\left(Z_{i,x_i}\right)\begin{pmatrix}1 & 0 & 0\end{pmatrix}^T = \left[Z_{i,x_i}\right]_{3,1}$

(b) $i = 1$, $x_1 = x_{2n} = x_{2n+1} = 0$. We get $\begin{pmatrix}0 & 0 & 1\end{pmatrix}\left(Z_{i,x_i}\right)\left(P_3\right)^2\begin{pmatrix}1 & 0 & 0\end{pmatrix}^T = \left[Z_{i,x_i}\right]_{1,1}$

(c) $i \in [3, 2n-1]$, $x_1 = 0$. We get $\begin{pmatrix}0 & 0 & 1\end{pmatrix}\left(P_3\right)\left(Z_{i,x_i}\right)\left(P_3\right)^2\begin{pmatrix}1 & 0 & 0\end{pmatrix}^T = \left[Z_{i,x_i}\right]_{1,2}$

(d) $i = 2n$, $x_1 = 0$. We get $\begin{pmatrix}0 & 0 & 1\end{pmatrix}\left(P_3\right)\left(Z_{i,x_i}\right)\left(P_3\right)\begin{pmatrix}1 & 0 & 0\end{pmatrix}^T = \left[Z_{i,x_i}\right]_{1,1}$

(e) $i = 2n+1$, $x_1 = 0$. We get $\begin{pmatrix}0 & 0 & 1\end{pmatrix}\left(P_3\right)^2\left(Z_{i,x_i}\right)\left(P_3\right)\begin{pmatrix}1 & 0 & 0\end{pmatrix}^T = \left[Z_{i,x_i}\right]_{2,2}$

(f) $i \in [2n+2, 3n]$, $x_1 = 0$. We get $\begin{pmatrix}0 & 0 & 1\end{pmatrix}\left(P_3\right)^3\left(Z_{i,x_i}\right)\begin{pmatrix}1 & 0 & 0\end{pmatrix}^T = \left[Z_{i,x_i}\right]_{3,1}$

Finally, define

$$\gamma_{i,x_i,parity(x_1)} = \begin{cases} \left[Z_{i,x_i}\right]_{3,1} & \text{if } parity(x_1) = 1 \\ \left[Z_{i,x_i}\right]_{1,1} & \text{if } parity(x_1) = 0,\ i = 0 \\ \left[Z_{i,x_i}\right]_{1,2} & \text{if } parity(x_1) = 0,\ i \in [1, 2n-1] \\ \left[Z_{i,x_i}\right]_{2,2} & \text{if } parity(x_1) = 0,\ i = 2n \\ \left[Z_{i,x_i}\right]_{2,3} & \text{if } parity(x_1) = 0,\ i = 2n+1 \\ \left[Z_{i,x_i}\right]_{3,1} & \text{if } parity(x_1) = 0,\ i \in [2n+2, 3n] \end{cases}$$

From here,

$$W_x = W_T = \sum_{i=1}^{3n}\Big(\cdots A_{i-1,x_{inp(i-1)}} \cdot Z_{i,x_{inp(i)}} \cdot A_{i+1,x_{inp(i+1)}} \cdots\Big) = \sum_{i=1}^{3n}\gamma_{i,x_i,parity(x_1)}$$

We have $p_T^{(1)} = U_T W_T$.

First, we claim that the following polynomial, taken from MSZ16, annihilates in the identity case:

Claim 3.3. *The following polynomial is identically zero when evaluated on handles from the all-identity branching program.*

$$\begin{aligned} Q_{1,2,3} = \ & \left(p_0^{(1)}p_{1,2,3}^{(1)}\right)^2 + \left(p_1^{(1)}p_{2,3}^{(1)}\right)^2 + \left(p_2^{(1)}p_{1,3}^{(1)}\right)^2 + \left(p_3^{(1)}p_{1,2}^{(1)}\right)^2 \\ & - 2\left(p_0^{(1)}p_{1,2,3}^{(1)}p_1^{(1)}p_{2,3}^{(1)} + p_0^{(1)}p_{1,2,3}^{(1)}p_2^{(1)}p_{1,3}^{(1)} + p_0^{(1)}p_{1,2,3}^{(1)}p_3^{(1)}p_{1,2}^{(1)}\right) \\ & - 2\left(p_1^{(1)}p_{2,3}^{(1)}p_2^{(1)}p_{1,3}^{(1)} + p_1^{(1)}p_{2,3}^{(1)}p_3^{(1)}p_{1,2}^{(1)} + p_2^{(1)}p_{1,3}^{(1)}p_3^{(1)}p_{1,2}^{(1)}\right) \\ & + 4\left(p_0^{(1)}p_{1,2}^{(1)}p_{1,3}^{(1)}p_{2,3}^{(1)} + p_{1,2,3}^{(1)}p_1^{(1)}p_2^{(1)}p_3^{(1)}\right) \end{aligned}$$

Proof. For a quick proof, note that the U_T, V_T for the all-identity case are the same for the 3-by-3 case as they are for the 2-by-2 case, and because all equations satisfied by the p_T are generated by the U_T, V_T, the fact that it annihilates in the 2-by-2 case shows that it must similarly hold in the 3-by-3 case.

The full proof is done by just exhaustively listing out the U_T and V_T and evaluating the polynomial. Here are all the U_T and V_T for various T:

T	U_T	V_T
$\emptyset$	U_0	V_0
$\{1\}$	U_1	V_1
$\{2\}$	U_2	V_2
$\{3\}$	U_3	V_3
$\{1,2\}$	U_1U_2/U_0	$V_1+V_2-V_0$
$\{2,3\}$	U_2U_3/U_0	$V_2+V_3-V_0$
$\{1,3\}$	U_1U_3/U_0	$V_1+V_3-V_0$
$\{1,2,3\}$	$U_1U_2U_3/U_0^2$	$V_1+V_2+V_3-2V_0$

Plugging these into $Q_{1,2,3}$ yields

$$\begin{aligned}
Q_{1,2,3} = \;& \big(p_0^{(1)}p_{1,2,3}^{(1)}\big)^2 + \big(p_1^{(1)}p_{2,3}^{(1)}\big)^2 + \big(p_2^{(1)}p_{1,3}^{(1)}\big)^2 + \big(p_3^{(1)}p_{1,2}^{(1)}\big)^2 \\
& - 2\big(p_0^{(1)}p_{1,2,3}^{(1)}p_1^{(1)}p_{2,3}^{(1)} + p_0^{(1)}p_{1,2,3}^{(1)}p_2^{(1)}p_{1,3}^{(1)} + p_0^{(1)}p_{1,2,3}^{(1)}p_3^{(1)}p_{1,2}^{(1)}\big) \\
& - 2\big(p_1^{(1)}p_{2,3}^{(1)}p_2^{(1)}p_{1,3}^{(1)} + p_1^{(1)}p_{2,3}^{(1)}p_3^{(1)}p_{1,2}^{(1)} + p_2^{(1)}p_{1,3}^{(1)}p_3^{(1)}p_{1,2}^{(1)}\big) \\
& + 4\big(p_0^{(1)}p_{1,2}^{(1)}p_{1,3}^{(1)}p_{2,3}^{(1)} + p_{1,2,3}^{(1)}p_1^{(1)}p_2^{(1)}p_3^{(1)}\big) \\
= \;& \Big(\big(U_0V_0\big)\big((U_1U_2U_3/U_0^2)(V_1+V_2+V_3-2V_0)\big)\Big)^2 \\
& + \Big(\big(U_1V_1\big)\big((U_2U_3/U_0)(V_2+V_3-V_0)\big)\Big)^2 \\
& + \Big(\big(U_2V_2\big)\big((U_1U_3/U_0)(V_1+V_3-V_0)\big)\Big)^2 \\
& + \Big(\big(U_3V_3\big)\big((U_1U_2/U_0)(V_1+V_2-V_0)\big)\Big)^2 \\
& - 2\Big(\big(U_0V_0\big)\big((U_1U_2U_3/U_0^2)(V_1+V_2+V_3-2V_0)\big)\big(U_1V_1\big)\big((U_2U_3/U_0)(V_2+V_3-V_0)\big)\Big) \\
& - 2\Big(\big(U_0V_0\big)\big((U_1U_2U_3/U_0^2)(V_1+V_2+V_3-2V_0)\big)\big(U_2V_2\big)\big((U_1U_3/U_0)(V_1+V_3-V_0)\big)\Big)
\end{aligned}$$

$$
\begin{aligned}
&- 2\Big(\big(U_0V_0\big)\big((U_1U_2U_3/U_0^2)(V_1+V_2+V_3-2V_0)\big)\big(U_3V_3\big)\big((U_1U_2/U_0)(V_1+V_2-V_0)\big)\Big)\\
&- 2\Big(\big(U_1V_1\big)\big((U_2U_3/U_0)(V_2+V_3-V_0)\big)\big(U_2V_2\big)\big((U_1U_3/U_0)(V_1+V_3-V_0)\big)\Big)\\
&- 2\Big(\big(U_1V_1\big)\big((U_2U_3/U_0)(V_2+V_3-V_0)\big)\big(U_3V_3\big)\big((U_1U_2/U_0)(V_1+V_2-V_0)\big)\Big)\\
&- 2\Big(\big(U_2V_2\big)\big((U_1U_3/U_0)(V_1+V_3-V_0)\big)\big(U_3V_3\big)\big((U_1U_2/U_0)(V_1+V_2-V_0)\big)\Big)\\
&+ 4\Big(\big(U_0V_0\big)\big((U_1U_2/U_0)(V_1+V_2-V_0)\big)\big((U_1U_3/U_0)(V_1+V_3-V_0)\big)\big((U_2U_3/U_0)(V_2+V_3-V_0)\big)\Big)\\
&+ 4\Big(\big((U_1U_2U_3/U_0^2)(V_1+V_2+V_3-2V_0)\big)\big(U_1V_1\big)\big(U_2V_2\big)\big(U_3V_3\big)\Big)\\
=\ &\Big((U_1U_2U_3V_0/U_0)(V_1+V_2+V_3-2V_0)\Big)^2\\
&+ \Big((U_1U_2U_3V_1/U_0)(V_2+V_3-V_0)\Big)^2\\
&+ \Big((U_1U_2U_3V_2/U_0)(V_1+V_3-V_0)\Big)^2\\
&+ \Big((U_1U_2U_3V_3/U_0)(V_1+V_2-V_0)\Big)^2\\
&- 2\Big((U_1U_2U_3V_0/U_0)(V_1+V_2+V_3-2V_0)(U_1U_2U_3V_1/U_0)(V_2+V_3-V_0)\Big)\\
&- 2\Big((U_1U_2U_3V_0/U_0)(V_1+V_2+V_3-2V_0)(U_1U_2U_3V_2/U_0)(V_1+V_3-V_0)\Big)\\
&- 2\Big((U_1U_2U_3V_0/U_0)(V_1+V_2+V_3-2V_0)(U_1U_2U_3V_3/U_0)(V_1+V_2-V_0)\Big)\\
&- 2\Big((U_1U_2U_3V_1/U_0)(V_2+V_3-V_0)(U_1U_2U_3V_2/U_0)(V_1+V_3-V_0)\Big)\\
&- 2\Big((U_1U_2U_3V_1/U_0)(V_2+V_3-V_0)(U_1U_2U_3V_3/U_0)(V_1+V_2-V_0)\Big)\\
&- 2\Big((U_1U_2U_3V_2/U_0)(V_1+V_3-V_0)(U_1U_2U_3V_3/U_0)(V_1+V_2-V_0)\Big)\\
&+ 4\Big((U_1U_2V_0)(V_1+V_2-V_0)(U_1U_3/U_0)(V_1+V_3-V_0)(U_2U_3/U_0)(V_2+V_3-V_0)\Big)\Big)\\
&+ 4\Big((U_1U_2U_3/U_0^2)(V_1+V_2+V_3-2V_0)(U_1U_2V_1V_2)(U_3V_3)\Big)\\
=\ &\Big(U_1U_2U_3/U_0\Big)\\
&\quad * \Big[((V_0)(V_1+V_2+V_3-2V_0))^2+\\
&\qquad + ((V_1)(V_2+V_3-V_0))^2 + ((V_2)(V_1+V_3-V_0))^2 + ((V_3)((V_1+V_2-V_0)))^2\\
&\qquad - 2\Big((V_0)(V_1+V_2+V_3-2V_0)(V_1)(V_2+V_3-V_0)\\
&\qquad\quad + (V_0)(V_1+V_2+V_3-2V_0)(V_2)(V_1+V_3-V_0)\\
&\qquad\quad + (V_0)(V_1+V_2+V_3-2V_0)(V_3)((V_1+V_2-V_0))\Big)
\end{aligned}
$$

$$
\begin{aligned}
&\quad - 2((V_1)(V_2+V_3-V_0)(V_2)(V_1+V_3-V_0)\\
&\qquad + (V_1)(V_2+V_3-V_0)(V_3)(V_1+V_2-V_0)\\
&\qquad + (V_2)(V_1+V_3-V_0)(V_3)(V_1+V_2-V_0))\\
&\quad + 4\Big((V_0)((V_1+V_2-V_0))(V_1+V_3-V_0)(V_2+V_3-V_0)\\
&\qquad + (V_1+V_2+V_3-2V_0)(V_1)(V_2)(V_3)\Big)\Big]\\
=\ &\Big(U_1U_2U_3/U_0\Big)\\
&\quad * \Big[\Big(4V_0^4 - 4V_0^3V_1 - 4V_0^3V_2 - 4V_0^3V_3 + 2V_0^2V_1^2 + 2V_0^2V_1V_2 + 2V_0^2V_1V_3 + 2V_0^2V_2^2\\
&\qquad + 2V_0^2V_2V_3 + 2V_0^2V_3^2 - 2V_0V_1^2V_2 - 2V_0V_1^2V_3 - 2V_0V_1V_2^2 - 2V_0V_1V_3^2 - 2V_0V_2^2V_3\\
&\qquad - 2V_0V_2V_3^2 + 2V_1^2V_2^2 + 2V_1^2V_2V_3 + 2V_1^2V_3^2 + 2V_1V_2^2V_3 + 2V_1V_2V_3^2 + 2V_2^2V_3^2\Big)\\
&\quad + \Big(-4V_0^3V_1 - 4V_0^3V_2 - 4V_0^3V_3 + 2V_0^2V_1^2 + 12V_0^2V_1V_2 + 12V_0^2V_1V_3\\
&\qquad + 2V_0^2V_2^2 + 12V_0^2V_2V_3 + 2V_0^2V_3^2 - 4V_0V_1^2V_2 - 4V_0V_1^2V_3 - 4V_0V_1V_2^2\\
&\qquad - 12V_0V_1V_2V_3 - 4V_0V_1V_3^2 - 4V_0V_2^2V_3 - 4V_0V_2V_3^2\Big)\\
&\quad + \Big(-2V_0^2V_1V_2 - 2V_0^2V_1V_3 - 2V_0^2V_2V_3 + 2V_0V_1^2V_2 + 2V_0V_1^2V_3\\
&\qquad + 2V_0V_1V_2^2 + 12V_0V_1V_2V_3 + 2V_0V_1V_3^2 + 2V_0V_2^2V_3 + 2V_0V_2V_3^2\\
&\qquad - 2V_1^2V_2^2 - 6V_1^2V_2V_3 - 2V_1^2V_3^2 - 6V_1V_2^2V_3 - 6V_1V_2V_3^2 - 2V_2^2V_3^2\Big)\\
&\quad + \Big(-4V_0^4 + 8V_0^3V_1 + 8V_0^3V_2 + 8V_0^3V_3 - 4V_0^2V_1^2 - 12V_0^2V_1V_2 - 12V_0^2V_1V_3\\
&\qquad - 4V_0^2V_2^2 - 12V_0^2V_2V_3 - 4V_0^2V_3^2 + 4V_0V_1^2V_2 + 4V_0V_1^2V_3 + 4V_0V_1V_2^2\\
&\qquad + 4V_0V_1V_3^2 + 4V_0V_2^2V_3 + 4V_0V_2V_3^2 + 4V_1^2V_2V_3 + 4V_1V_2^2V_3 + 4V_1V_2V_3^2\Big)\Big]\\
=\ &0
\end{aligned}
$$

□

Next, we claim that it does not annihilate in the anti-identity case.

Claim 3.4. *Let $Q_{1,2,3}$ be defined as above. Then $Q_{1,2,3}$ is identically non-zero when evaluated on handles from the permutation branching program.*

Proof. As with before, the proof is done via direct computation. Recall that we have

$$
W_T = \begin{cases} \sum [Z_{i,x_i}]_{3,1} & \text{if } 1 \in T \\ [Z_{1,0}]_{1,1} + \sum_{i=2}^{2n-1} [Z_{i,x_i}]_{1,2} + [Z_{2n,0}]_{2,2} & \\ \quad + [Z_{2n+1,0}]_{2,3} + \sum_{i=2n+2}^{3n} [Z_{i,x_i}]_{3,1} & \text{otherwise} \end{cases}
$$

For the U_T, we have

T	U_T
$\emptyset$	U_0
$\{1\}$	U_1
$\{2\}$	U_2
$\{3\}$	U_3
$\{1,2\}$	U_1U_2/U_0
$\{2,3\}$	U_2U_3/U_0
$\{1,3\}$	U_1U_3/U_0
$\{1,2,3\}$	$U_1U_2U_3/U_0^2$

Note that the U_T remain the same as before. Thus, as with before, all the U_T factor out and we are left with

$$\begin{aligned} Q_{1,2,3} = \ & \Big(U_1U_2U_3/U_0\Big) \\ & * \Big[\big(W_0W_{1,2,3}\big)^2 + \big(W_1W_{2,3}\big)^2 + \big(W_2W_{1,3}\big)^2 + \big(W_3W_{1,2}\big)^2 \\ & \quad - 2\big(W_0W_{1,2,3}W_1W_{2,3} + W_0W_{1,2,3}W_2W_{1,3} + W_0W_{1,2,3}W_3W_{1,2}\big) \\ & \quad - 2\big(W_1W_{2,3}W_2W_{1,3} + W_1W_{2,3}W_3W_{1,2} + W_2W_{1,3}W_3W_{1,2}\big) \\ & \quad + 4\big(W_0W_{1,2}W_{1,3}W_{2,3} + W_{1,2,3}W_1W_2W_3\big)\Big] \end{aligned}$$

Now we consider the term $z := [Z_{2,0}]^2_{3,1}$. This only shows up in W_T where $1 \in T$ and $2 \notin T$. If we look at our evaluation of $Q_{1,2,3}$, we see that only three terms contain a factor of z^2: $(W_1W_{2,3})^2$, $(W_2W_{1,3})^2$, and $-2(W_1W_{2,3}W_2W_{1,3})$. These yield a coefficient of

$$(W_{2,3})^2 + (W_2)^2 - 2(W_{2,3}W_2) = (W_{2,3} - W_2)^2 = 0 \iff W_{2,3} = W_2.$$

But with high probability $W_{2,3} \neq W_2$, thus $Q_{1,2,3}$ does not annihilate in the permutation case.

□

To conclude, here is an overview of the attack. First, we fix all but the first three bits on the input x, and vary the first three bits over all subsets of [3]. Submitting these inputs to **Type 1** queries gives handles to elements $p_T^{(0)}$, with $T \subseteq \{1,2,3\}$. This allows us to evaluate $Q_{1,2,3}$, and if the result is 0, then we are in the all-identity case, and otherwise, we are in the permutation case.

3.3 Attacking GGH

Introduction. Now we adapt the attack given above to GGH encodings. The only tweak we need to make is to account for the fact that "public" variables no longer look like $Y_i = q_i(\{X_j\}) + gZ_i$, rather, they look like $Y_i = \frac{q_i(\{X_j\}) + gZ_i}{z^j}$ for some $j \leq k$, the multilinearity parameter. Additionally, an execution of the program will yield an encoding

$$p_x = \left[\left(p_x^{(0)}(\{X_j\}, \{Z_i\}) + gp_x^{(1)}(\{X_j\}, \{Z_i\}) + g^2 ...\right)/z^k\right]_q$$

As before, we know that $p_x^{(0)}(\{X_j\}, \{Z_i\}) = 0$. We can thus implement a **Type 1** query by multiplying this by the zero-test parameter $p_{zt} = [hz^k/g]_q$, giving

$$W_x := [p_x \cdot p_{zt}]_q = h \cdot \left(p_x^{(1)}(\{X_j\}, \{Z_i\}) + gp_x^{(2)}(\{X_j\}, \{Z_i\}) + g^2 ...\right)$$

The difference now is that we have an extra factor of h. Since our annihilating polynomial is homogenous, it will annihilate $p_x^{(1)}$ if and only if it also annihilates $h \cdot p_x^{(1)}$ (for $h \neq 0$).

Now, in order to check whether a polynomial is an annihilating polynomial, notice that if the $p_x^{(1)}$ are annihilated, we are left with a ring element in the ideal $\langle hg \rangle$. Thus, the goal will be to compute a basis for $\langle hg \rangle$ by calculating a number of W_x via **Type 1** queries and then annihilating with $Q_{1,2,3}$. Finally, to implement the **Type 2** query, calculate one more element which is either in $\langle hg \rangle$ or not, depending on which program was obfuscated.

Attack. Our two branching programs A, A' are the same as before. In particular, recall that for the permutation case,

$$A'_{1,0} = A'_{2n,0} = A'_{2n+1} \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}.$$

All other matrices are the 3-by-3 identity.

One crucial observation is that we can take any subset $T_0 \subseteq [2, n]$ with $|T_0| = 3$. Then, if we pick some fixing on the bits outside of T_0, while varying the bits in T_0, the annihilating polynomial $Q_{1,2,3}$ still annihilates the handles returned by the **Type 1** queries in both cases. This is because these handles all obey the same U_T and V_T equations as before.

Thus, we will use this to create a basis for the ideal $\langle hg \rangle$ regardless of which branching program we are evaluating. Then, we evaluate the program while varying bits from a subset of $[n]$ that include 1. The permutation branching program should output something that falls outside the ideal $\langle hg \rangle$, since this evaluation includes the permutation matrices, while the identity branching program should still annihilate.

4 Limitations of annihilation attacks for computationally bounded adversaries

In this section, we describe a brute-force approach to finding annihilation polynomials that fails for anything but the trivial branching program. Specifically, we were unable to find an annihilation polynomial even for the anti-identity case given in MSZ16, which is the simplest branching program after the all-identity case, running on a single core for several days. By comparison, the annihilation polynomial for the all-identity case was found after a few hours. Though entirely based on observation, this computational blow-up is not promising for practical implementations of the annihilation attack.

4.1 Complexity of a brute force search

The brute force approach takes two parameters. First is the number of input bits that are allowed to vary, and second is the degree of the annihilation polynomial. Briefly, by increasing the number of input bits that can vary, we get more handles $p_x^{(1)}$, and thus more potential relations between the handles to aid in the annihilation; by increasing the degree of the annihilation polynomial, we gain the ability to exploit higher-order relations between handles.

The brute force approach merely constructs every possible term of a certain degree from the given handles and derives constraints on the coefficients of each term. Note that we are given that the polynomial must be homogenous, in order to break the obfuscation scheme of GGH13.

Unfortunately, the size of the search space is exponential in the number of variables. In particular, it is well known that the number of terms of degree d in n variables is

$$\frac{(d+n-1)!}{d!(n-1)!} = \binom{d+n-1}{d}$$

This can be shown from the familiar "stars-and-bars" argument. Then, for fixed n, we have

$$\binom{d+n-1}{d} \geq \left(\frac{d+n-1}{d}\right)^d \underset{d\to\infty}{\to} e^{n-1}$$

where e is Euler's constant.

To complicate things even further, the number of variables n is exponential in the number of bits to vary: for m bits, we have $n = 2^m$, so in particular, the number of terms (and thus coefficients) as a function of the number of bits to vary m when we send the degree of the polynomial to infinity is

$$e^{2^m-1} \sim e^{2^m}$$

Thus, we end up solving a system of linear constraints in a super-exponential number of variables, and without a bound on the necessary number of bits to vary or a more clever way to derive constraints given the structure of the branching program, the search space quickly gets out of hand.

4.2 Straddling sets

Some constructions of obfuscation drop the $\alpha_{i,b}$ and instead force the $Z_{i,b}$ to come from so-called straddling sets. For the purposes of this section, we needn't worry about the extra structure coming from the straddling sets, since for this attack, the $Z_{i,b}$ are just free variables.

Notice however that the handles returned by **Type 1** query is now just:

$$p_x^{(1)} = A_0 \cdot \sum_{i=1}^{n} \Big(\cdots A_{i-1,x_{i-1}} \cdot Z_{i,x_i} \cdot A_{i+1,x_{i+1}} \cdots \Big) \cdot A_{n+1}.$$

In particular, this expression is linear in the $Z_{i,b}$, and thus, we can restrict our search to polynomials of degree 1. In this case, the brute force attack, for n variables, has only

$$\binom{d+n-1}{d} = \binom{1+n-1}{1} = n-1$$

variables, yielding

$$2^m - 1 \sim 2^m$$

constraints. While this is still exponential, it seems possible that restricting the search space to linear polynomials opens up many more optimization techniques. Regardless, the idea that the $\alpha_{i,b}$ are unnecessary seems to be at least somewhat untrue, in that they greatly increase the complexity of the annihilation attack.

4.3 Some examples without alphas

MSZ16 gives some linear constraints for the anti-identity branching program without alphas. Recall from section 2.1 the definition of the anti-identity branching program. Briefly, the program has $2n+2$ layers. The bookend matrices are the vectors $A_0 := \begin{pmatrix} 0 & 1 \end{pmatrix}$ and $A_{2n+1} := \begin{pmatrix} 1 & 0 \end{pmatrix}^T$. The middle $2n$ layers scan each input bit twice, once going forward then once going backward, so that the input selection function $inp(i) := \min(i, 2n+1-i)$. Additionally, the matrix at every layer reading input bit 1 when the input bit is 0 is the 2-by-2 anti-diagonal matrix, while the case where the input bit is 1 is the 2-by-2 identity matrix. In other words,

$$A'_{1,0} = A'_{2n,0} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

All other matrices are the identity matrix. Clearly this branching program always evaluates to 0.

As before, we define W_T to be the handle returned in response to a **Type 1** query on a set that is 1 in positions $i \in T$ and 0 everywhere else. Then taking $T \subseteq \{1, 2, 3\}$, MSZ16 derive the following constraints:

$$W_{1,2,3} + W_1 = W_{1,2} + W_{1,3}$$
$$W_{2,3} + W_0 = W_2 + W_3$$

We stress that without the alphas, these would be annihilating polynomials.

As a second example, let us consider the 3-by-3 permutation branching program given in Section 3.1. Recall that the program has $3n + 2$ layers, with bookend vectors $A_0 := \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}$ and $A_{3n+2} := \begin{pmatrix} 1 & 0 & 0 \end{pmatrix}^T$. The middle $3n$ layers scan each input bit three times, once going forward, once going backward, and one more time forwards, so that the input selection function is

$$inp(i) := \begin{cases} i & 0 \leq i \leq n \\ 2n + 1 - i & n + 1 \leq i \leq 2n \\ i - 2n & 2n + 1 \leq i \leq 3n \end{cases}$$

Additionally, the matrix at every layer reading input bit 1 when the input bit is 0 is a 3-by-3 permutation matrix, while the case where the input bit is 1 is the 3-by-3 identity matrix. In other words,

$$A'_{1,0} = A'_{2n,0} = A'_{2n+1} \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

As before, all other matrices are the identity matrix, and this branching program always evaluates to 0.

Recall from Claim 3.4 that we have the following handles returned on **Type 1** queries:

$$W_T = \begin{cases} \sum [Z_{i,x_i}]_{3,1} & \text{if } 1 \in T \\ [Z_{1,0}]_{1,1} + \sum_{i=2}^{2n-1} [Z_{i,x_i}]_{1,2} + [Z_{2n,0}]_{2,2} & \\ \quad + [Z_{2n+1,0}]_{2,3} + \sum_{i=2n+2}^{3n} [Z_{i,x_i}]_{3,1} & \text{otherwise} \end{cases}$$

Here, the $Z_{i,b}$ are the free variables we want to annihilate. We will attempt to simply

the problem by creating new variables that are just linear combinations of these $Z_{i,b}$.

$$\gamma_{1,0} = [Z_{1,0}]_{1,1} + [Z_{2n,0}]_{1,1} + [Z_{2n+1,0}]_{1,1} + \sum_{i=4}^{2n-3} [Z_{i,0}]_{3,1}$$
$$\gamma_{1,1} = [Z_{1,0}]_{3,1} + [Z_{2n,0}]_{3,1} + [Z_{2n+1,0}]_{3,1} + \sum_{i=4}^{2n-3} [Z_{i,1}]_{1,2}$$

$$\gamma_{2,0,0} = [Z_{2,0}]_{1,2} + [Z_{2n-1,0}]_{1,2} + [Z_{2n+2,0}]_{3,1}$$
$$\gamma_{2,0,1} = [Z_{2,0}]_{3,1} + [Z_{2n-1,0}]_{3,1} + [Z_{2n+2,0}]_{3,1}$$
$$\gamma_{2,1,0} = [Z_{2,1}]_{1,2} + [Z_{2n-1,1}]_{1,2} + [Z_{2n+2,1}]_{3,1}$$
$$\gamma_{2,1,1} = [Z_{2,1}]_{3,1} + [Z_{2n-1,1}]_{3,1} + [Z_{2n+2,1}]_{3,1}$$

$$\gamma_{3,0,0} = [Z_{3,0}]_{1,2} + [Z_{2n-2,0}]_{1,2} + [Z_{2n+3,0}]_{3,1}$$
$$\gamma_{3,0,1} = [Z_{3,0}]_{3,1} + [Z_{2n-2,0}]_{3,1} + [Z_{2n+3,0}]_{3,1}$$
$$\gamma_{3,1,0} = [Z_{3,1}]_{1,2} + [Z_{2n-2,1}]_{1,2} + [Z_{2n+3,1}]_{3,1}$$
$$\gamma_{3,1,1} = [Z_{3,1}]_{3,1} + [Z_{2n-2,1}]_{3,1} + [Z_{2n+3,1}]_{3,1}$$

$$\gamma_0 = \sum_{i=2n+4}^{3n} [Z_{i,0}]_{3,1}$$

Simple inspection reveals that the following simplifications:

$$W_0 = \gamma_{1,0} + \gamma_{2,0,0} + \gamma_{3,0,0} + \gamma_0$$
$$W_1 = \gamma_{1,1} + \gamma_{2,0,1} + \gamma_{3,0,1} + \gamma_0$$
$$W_2 = \gamma_{1,0} + \gamma_{2,1,0} + \gamma_{3,0,0} + \gamma_0$$
$$W_3 = \gamma_{1,0} + \gamma_{2,0,0} + \gamma_{3,1,0} + \gamma_0$$
$$W_{1,2} = \gamma_{1,1} + \gamma_{2,1,1} + \gamma_{3,0,1} + \gamma_0$$
$$W_{1,3} = \gamma_{1,1} + \gamma_{2,0,1} + \gamma_{3,1,1} + \gamma_0$$
$$W_{2,3} = \gamma_{1,0} + \gamma_{2,1,0} + \gamma_{3,1,0} + \gamma_0$$
$$W_{1,2,3} = \gamma_{1,1} + \gamma_{2,1,1} + \gamma_{3,1,1} + \gamma_0$$

From here, it is easy to verify that these are the only linear constraints satisfied by the W_T:

$$W_{1,2,3} + W_1 = W_{1,2} + W_{1,3}$$
$$W_{2,3} + W_0 = W_2 + W_3$$

5 Limitations of annihilation attacks for computationally unbounded adversaries

In this section we characterize certain pairs of branching programs that cannot be differentiated via an annihilation attack, even for computationally unbounded adversaries. The motivation lies in the fact that, so far, the single known construction of an annihilation polynomial is only able to positively identify the trivial branching program. Indeed, one proof of the fact that polynomial does not annihilate in the anti-identity case relies on a dimensionality argument. We seek to explore more carefully how the complexity of the branching program factors into annihilation attacks.

The central claim is that if the two branching programs consist of the matrices $A_{i,b}$ and $A'_{i,b}$ for $i \in [l]$ and $b \in \{0,1\}$, and there exist one matrix P such that $A_{i,b} = PA'_{i,b}P^{-1}$ for all $i \in [l]$ and $b \in \{0,1\}$ and $A_0 = A'_0P^{-1}$, $A_{n+1} = PA'_{n+1}$, then the annihilation attack will always fail. This captures in some sense the intuition that if the algebraic spaces that the two programs inhabit are somewhat "similar enough," then the annihilation attack cannot distinguish between the two programs by the algebraic manipulation of handles.

The statement, though simple to prove on its own, turns out to have interesting implications when interpreting branching programs as finite state automota. First, we give an illustrative example for branching programs consisting of only permutation matrices and bookend vectors that are 1 in one position, and 0 everywhere else. Then, we drop the requirement that the branching programs consist of permutation matrices, and relax the bookend vectors.

5.1 Setting

For this attack, we use the definition of a branching program given in MSZ16. In particular, recall that a branching program of length $l+2$ consists of $2l$ n-by-n matrices $A_{i,b}$ and $A'_{i,b}$ for $i \in [l]$ and $b \in \{0,1\}$ and two bookend vectors, A_0 and A_{n+1} that are n-by-1 and 1-by-n, respectively. To simplify notation, we only consider branching programs for the input function $inp(i) = i$; the proofs can be easily generalized to arbitrary input functions.

We evaluate the branching program on an input x of length l by taking the product:

$$A_0 \cdot \left(\prod_{i=1}^{n} A_{i,x_i}\right) \cdot A_{n+1}$$

The program returns 0 if and only if this product evaluates to 0; it returns 1 otherwise. For a more complete definition, plus the obfuscation scheme, we refer the reader to MSZ16 (or the first section of this paper).

5.2 Proof

We begin with a statement of the claim.

Claim 5.1. *Assume we have a pair of branching programs $\mathcal{P}$ and $\mathcal{P}'$, each of length l, given by n-by-n matrices $A_{i,b}$ and $A'_{i,b}$ for $i \in [l]$ and $b \in \{0,1\}$, n-by-1 bookend vectors A_0 and A'_0, and 1-by-n bookend vectors A_{n+1} and A'_{n+1}, respectively. If there exists some invertible matrix P such that*

$$\begin{aligned} A_{i,b} &= PA'_{i,b}P^{-1},\ i \in [l], b \in \{0,1\} \\ A_0 &= A'_0P^{-1} \\ A_{n+1} &= PA'_{n+1} \end{aligned}$$

then the annihilation attack given in MSZ16 will be unable to distinguish the two obfuscated branching programs.

Proof. Assume first that the branching programs are un-obfuscated. It is easy to see that in order to evaluate the branching programs on some input x, we have

$$\begin{aligned} \mathcal{P}(x) &= A_0 \cdot \big(\prod_{i=1}^{n} A_{i,x_i}\big) \cdot A_{n+1} \\ &= A_0P^{-1} \cdot \big(\prod_{i=1}^{n} PA_{i,x_i}P^{-1}\big) \cdot PA_{n+1} \\ &= \mathcal{P}'(x). \end{aligned}$$

The proof for obfuscated programs is more involved, if only slightly so. The key observation is that to achieve equivalence under an annihilation polynomial, all we need to show is that the formal variables returned from a **Type 1** query is equivalent up to a "renaming" of the free variables.

Recall that a **Type 1** query returns a handle to the following variable:

$$p_x^{(1)} = \rho\Big(\prod_i \alpha_{i,x_i}\Big) \cdot A_0 \cdot \sum_{i=1}^{n}\Big(\cdots A_{i-1,x_{i-1}} \cdot Z_{i,x_i} \cdot A_{i+1,x_{i+1}} \cdots\Big) \cdot A_{n+1}.$$

Note that the only free variables are the α_{i,x_i} and Z_{i,x_i}.

As before, we associate an input $x \in \{0,1\}^n$ with a set $T \subseteq [n]$ where $i \in T \iff x_i = 1$ (we denote the empty set 0). Letting $\alpha'_{i,b} = \alpha_{i,b}\alpha_{2n+1-i,b}\alpha_{2n+i}$ for $i \in [2,n]$ and $\alpha'_{1,b} = \rho\alpha_{1,b}\alpha_{2n,b}\alpha_{2n+1}$ yields

$$U_x = U_T = \textstyle\prod_i \alpha'_{i,x_i} = \rho \prod_i \alpha_{i,x_i}$$

which satisfies the following identity:

$$U_T = \prod_{S \subseteq T, |S| \leq 1} U_S^{\left[\binom{|T|-|S|-1}{1-|S|}(-1)^{|S|+1}\right]}$$

Importantly, this identity comes from the obfuscation scheme and is *independent of the structure of the branching program.* Thus, it is equivalent for differing branching programs up to a renaming of variables. In other words, if we can prove similar equivalence up to a renaming of variables on the second half of $p_x^{(1)} = W_T$, then any annihilating polynomial on one branching program will always annihilate on the other as well.

But this is also follows directly from the evaluation of branching programs. Notice that

$$\begin{aligned} W_T &= A_0 \cdot \sum_{i=1}^{n} \Big(\cdots A_{i-1,x_{i-1}} \cdot Z_{i,x_i} \cdot A_{i+1,x_{i+1}} \cdots \Big) \cdot A_{n+1} \\ &= A_0 P^{-1} \cdot \sum_{i=1}^{n} \Big(\cdots \big(PA_{i-2,x_{i-2}} P^{-1}\big) \cdot \big(PA_{i-1,x_{i-1}}\big) \cdot Z_{i,x_i} \\ &\qquad \cdot \big(A_{i+1,x_{i+1}} P^{-1}\big) \cdot \big(PA_{i+2,x_{i+2}} P^{-1}\big) \cdots \Big) \cdot PA_{n+1} \\ &= A'_0 \cdot \sum_{i=1}^{n} \Big(\cdots A'_{i-1,x_{i-1}} \cdot \big(PZ_{i,x_i} P^{-1}\big) \cdot A'_{i+1,x_{i+1}} \cdots \Big) \cdot A'_{n+1}. \end{aligned}$$

The only free variables that need to be annihilated here are the Z_{i,x_i} and Z'_{i,x_i} (the A_{i,x_i} and A'_{i,x_i} are given by the branching programs). Thus, all we need to do is "rename" the Z'_{i,x_i} as $PZ_{i,x_i}P^{-1}$, and *all relations satisfied by the W_T are also satisfied by W'_T.* This completes the proof. □

5.3 A graph theoretical interpretation

Branching programs have well-known representations as finite state automata. This conversion into a graph has interesting implications for the claim proven above.

We begin with the un-obfuscated case. To begin, let the dimension of the matrices $A_{i,b}$ in the branching program be $n \times n$, and restrict these matrices to permutation matrices. Consider a graph on n vertices, labelled 1 to n. Define unit vectors e_j to be 1-by-n unit vectors that are 1 in the j^{th} position and 0 everywhere else, and restrict the bookend vectors to unit vectors (as is the setting in the MSZ16 attack). WLOG assume that the bookend vectors are $A_0 = e_1$ and $A_{n+1} = e_1^T$.

We define also a source and a sink node, for a total of $n + 2$ nodes in the graph. Then given $A_0 = e_i$, we add one edge directed from the source node to node i; in particular, we have an edge from the source node to node 1. Similarly, the second bookend vector $A_{n+1} = e_j$ defines an edge to sink node from node j – here, we again have node 1. Then each permutation matrix $A_{i,b}$ defines how to move this unit of flow along edges in the graph between nodes, and the branching program evaluates to 1 if the flow at the sink node is 1, and 0 otherwise.

Specifically, at layer $i \in [l]$ of the branching program, we create two new graphs $G_{i,b}$, corresponding to $A_{i,b}$ for $b \in \{0,1\}$ (denote the corresponding graphs for $A'_{i,b}$ as $G'_{i,b}$). For each bit b, we add a directed edge from node j to k iff $e_j A_{i,b} = e_k$. In particular, each graph should have n directed edges, including self edges, with exactly one edge directed out from each node. We see also that the $A_{i,b}$ and $A'_{i,b}$ are just adjacency matrices for the $G_{i,b}$ and $G'_{i,b}$, respectively.

It should be clear that to evaluate the branching program, we begin with 1 unit of flow at the source. Then, at each layer, we begin with 1 unit of flow where the previous layer stopped and send the flow along exactly one edge. For example, suppose layer i left 1 unit of flow at vertex j. Then for layer $i+1$, we begin with 1 unit of flow at vertex j and send it along the single edge directed out from j, say to k. Then layer $i+2$ begins with 1 unit of flow at vertex k, and so on, until the last layer, in which we check to see how much flow is at the sink node. The branching program then returns 0 if there is no flow at the sink node, and 1 otherwise.

Now, imagine that there is some labelling of vertices in the $G_{i,b}$ and $G'_{i,b}$, such that the numbering is consistent within the $\{G\}$ and the $\{G'\}$, and further, for all edges $e \in G_{i,b} \iff e \in G'_{i,b}$ for all i and b. Then clearly the evaluations of the flow under the G and G' are equivalent up to a renaming of the nodes. Furthermore, this renaming is given exactly by some permutation matrix P, i.e. $A_{i,b} = P A'_{i,b} P^T$ for all i and b. But we also have that, for permutation matrices, $P^{-1} = P^T$. This is exactly the same result as proven previously.

Next, in order to allow arbitrary matrices $A_{i,b}$, we allow for weighted edges, where again the $A_{i,b}$ is the usual adjacency matrix. Denote the weight of an edge from a node i to a node j as $w_{i,j}$, and the flow at a node i as f_i. To evaluate the flow at a particular layer, for every unit of flow at a node i and every edge directed out from node i to nodes j with weight $w_{i,j}$, we put $w_{i,j}$ flow at node w_j. In particular, the flow at a node j is $\sum_{i \to j} w_{i,j} f_i$, where the sum is taken over all nodes i with a directed edge from i to j.

Finally, to allow for arbitrary bookend vectors, we create n edges from the source (and n edges to the sink), where the weight on each edge is given by the bookend vectors.

5.4 The converse is false.

If the converse of some variation of Claim 5.1 were true, then we would have a complete description of which pairs of branching programs could be distinguished using an annihilation attack. Unfortunately, the way Claim 5.1 is currently formulated, the converse is definitely false.

In particular, notice that in Subsection 4.3, the two programs given satisfy the same linear constraints on their W_T. One can easily make the 2-by-2 anti-identity case into a 3-by-3 case as follows: Let the bookend matrices be $A_0 := \begin{pmatrix} 0 & 1 & 0 \end{pmatrix}$ and $A_{2n+1} := \begin{pmatrix} 1 & 0 & 0 \end{pmatrix}^T$. Let

$$A'_{1,0} = A'_{2n,0} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Let all other matrices are the identity matrix. It is easy to see that the new dimension does not affect the evaluation of the branching program whatsoever.

Though we have only explicitly calculated the W_T for $T = \{1, 2, 3\}$, it turns out that the W_T for both branching programs satisfy the same constraints for any T. This is because the way we have composed the free variables (i.e. the $Z_{i,b}$) into γ should always be analogous between the two branching programs. In particular, we can always group terms by dependencies:

1. $\gamma_{1,b}$ for $b \in \{0, 1\}$
   Terms that depend on whether $b = x_1$ is 1 or 0 (corresponding to whether $A_{i,b}$ is the identity or not). We abuse notation and let $\gamma_{1,b,b} := \gamma_{1,b}$.

2. $\gamma_{j,i,b}$ for $j \in T - \{1\}, i \in \{0, 1\}, b \in \{0, 1\}$
   Terms that depend on whether $b = x_1$ is 1 or 0, AND whether $i = x_j$ is 1 or 0.

3. γ_0
   Terms that do not depend on anything.

It is easy to derive the following equations for the W_T:

$$W_T = \sum_{j \in T} \gamma_{j,x_j,x_1} + \gamma_0$$

More generally, it seems that the following family of branching programs are indistinguishable via the annihilation attack:

Claim 5.2. *Let $\mathcal{P}$ and $\mathcal{P}'$ be two branching programs of the same length. Assume that the branching programs share the same bookend vectors, and that all matrices in the branching programs are the identity, except for one layer k (and k' respectively), where the matrices can be arbitrary invertible matrices. If this layer reads the same input bit for both branching programs, then $\mathcal{P}$ and $\mathcal{P}'$ are indistinguishable via annihilation attacks.*

Proof. It is easy to see that we can build γ as above for all the W_T. Assume that this implies that the W_T for both branching programs satisfy the same linear relationships. Since the α's from the obfuscation yield the same linear relationships regardless of branching program, the handles (which are products of the α's and γ's) must satisfy the same relationships. Thus any polynomial that annihilates for one branching program will also annihilate for the other.

The only thing that remains to be shown is that the W_T satisfy the same relationships for both branching programs. There are two ways this could be violated. First is if we have not fully characterized all the γ. In other words, the W_T cannot be fully expressed

as a sum of the γ constructed above. Recall that we have split the terms in the sum into 3 types: terms that depend on only what the input bit read at layer k is, terms that depend on what the input bit read at layer k is AND whether some $t \in T$ is 1 or 0, and finally terms that do not depend on anything. However, it should be clear that these are the only possible dependencies: given that only one bit in the input matters, along with the fact that we are restricting changes in input to bits in T, these are the only possible variables to depend on.

Second is if the γ were not independent, thus giving rise to extra relationships between the W_T that may be satisfied by one program, but not the other. The only way this would happen is if one of the $\gamma_{j,i,b}$ does not depend on either $b = x_k$ or $i = x_j$. Notice that this might happen in some cases where we allow more than one layer, say two layers l and m to depend on the input bit. Then it is possible that $A_{l,0} \cdot A_{m,0} = A_{l,1} \cdot A_{m,1}$, and so for certain terms it does not matter whether x_k is 0 or 1. However, since only one layer is reading the input bit, we can avoid this case. □

Note that Claim 5.2 is actually weaker than the examples given. In particular, the examples allow multiple layers to depend on a single bit, while Claim 5.2 only allows one layer to depend on a single bit. The complexity in proving the multiple layers version can be seen from the following example: assume that the branching program consists of the following layers: in layers 1 and 2, the matrix is anti-identity if the first input bit is 0, and they are identity otherwise. The rest of the layers are all identity. It should be clear, then, that most terms do not depend on the input bit at all. Specifically, when considering the sum

$$W_T = A_0 \cdot \sum_{i=1}^{n} \Big(\cdots A_{i-1,x_{i-1}} \cdot Z_{i,x_i} \cdot A_{i+1,x_{i+1}} \cdots \Big) \cdot A_{n+1},$$

only the terms for $i = 1, 2$ have any dependency on the input bit, and so we have that $\gamma_0 \neq 0$.

Compare this to the branching program for which the first and last layers are anti-identity if the first input bit is 0, and identity otherwise. The rest of the layers are again all identity. Then all terms depend on the first input bit. In fact, this is exactly the anti-identity branching program, and it turns out that $\gamma_0 = 0$.

As it turns out, this difference between the γ_0 term is not enough to show that the two can be differentiated via an annihilation attack. In particular, we have shown that the anti-identity can not be differentiated from the 3-by-3 permutation, even though the 3-by-3 permutation has non-zero γ_0. The reason is that it turns out that satisfying that the $\gamma_{1,b}$ terms are zero already forces the W_T to satisfy that the γ_0 term is 0. In particular, we have for the 3-by-3 permutation case that the $\gamma_{1,1}$ terms force

$$\sum_{T:1 \in T} W_T = 0,$$

while the the $\gamma_{1,0}$ terms give

$$\sum_{T:1\notin T} W_T = 0.$$

This yields that

$$\sum_{T} W_T = 0,$$

and since every W_T term has a single γ_0 term, we can just ignore the constraints on the γ_0. However, we may not always be so lucky, and in general, we have no way to guarantee that the extra constraints will be vacuously satisfied by some other common constraints.

Apart from allowing more layers to depend on a single input bit, the next step is to consider whether we can somehow get two layers depending on two separate input bits. For one, we will have a lot more γ's, because the number of combinations of variables each term could depend on is essentially doubled. However, this should not be a problem because the W_T for both branching programs are still the same in terms of the γ's. Let the positions for which switching the bits gives different matrices be denoted j and k, then we have the following set of γ's:

1. $\gamma_{j,b}$ for $b \in \{0,1\}$
   Terms that only depend on whether $b = x_j$ is 1 or 0 (corresponding to whether $A_{j,b}$ is the identity or not).

2. $\gamma_{k,b}$ for $b \in \{0,1\}$
   Terms that only depend on whether $b = x_k$ is 1 or 0 (corresponding to whether $A_{k,b}$ is the identity or not).

3. $\gamma_{i,b,j'}$ for $i \in T - \{j,k\}, b, j', k' \in \{0,1\}$
   Terms that depend on whether $b = x_i$ is 1 or 0, and whether $j' = x_j$ is 1 or 0.

4. $\gamma_{i,b,k'}$ for $i \in T - \{j,k\}, b, j', k' \in \{0,1\}$
   Terms that depend on whether $b = x_i$ is 1 or 0, and whether $k' = x_k$ is 1 or 0.

5. $\gamma_{i,b,j',k'}$ for $i \in T - \{j,k\}, b, j', k' \in \{0,1\}$
   Terms that depend on whether $b = x_i$ is 1 or 0, whether $j' = x_j$ is 1 or 0, and whether $k' = x_k$ is 1 or 0.

6. γ_0
   Terms that do not depend on anything.

As it turns out, if we require that $A_{j,b} \cdot A_{k,b'}! = I_n$ for $(b,b') \in \{0,1\}^2$, then we have no $\gamma_{j,b}$ or $\gamma_{k,b}$ terms. Further, notice that these terms are disjoint, so we can abuse notation and write $\gamma_{i,b,j',k} := \gamma_{i,b,j'}$ and $\gamma_{i,b,j,k'} := \gamma_{i,b,k'}$. Then we have the following expression for W_T:

$$W_T = \sum_{i \in T} \gamma_{i,x_i,x_j,x_k} + \gamma_0$$

Here is a claim characterizing this family of branching programs:

Claim 5.3. *Let $\mathcal{P}$ and $\mathcal{P}'$ be two branching programs of the same length. Assume that the branching programs share the same bookend vectors, and that all matrices in the branching programs are the identity, except for two layers j and k (j' and k' respectively), where the matrices are invertible matrices such that $A_{j,b} \cdot A_{k,b'}! = I_n$ for $(b, b') \in \{0,1\}^2$. If layers j and j' read the same input bits, and layers k and k' read the same input bits, then $\mathcal{P}$ and $\mathcal{P}'$ are indistinguishable via annihilation attacks.*

Proof. As before, we need to show two things. First, that the expression for W_T as a sum of the γ is correct, and second, that there are no "extra" relationships between the γ, i.e. they are independent variables. This follows almost immediately from the arguments in the proof of Claim 5.2. □

More generally, we can expand this notion to arbitrary numbers of layers by defining the following notions.

Definition 5.4 (Input Equivalent)**.** *We call two branching programs $\mathcal{P}$ and $\mathcal{P}'$ **input equivalent** if they are the same length, and for all layers that depend on input bits in $\mathcal{P}$, the same layers in $\mathcal{P}'$ also depend on those same input bits.*

Definition 5.5 (Strongly Unique)**.** *We say a branching program $\mathcal{P}$ of length n with input function inp is **strongly unique** if there is no subsequence of $\{A_{i,b}\}_{i=1}^n$ such that the product is the same for two different settings of b's. In other words, for any $j \geq$, $k \leq n$, and for all pairs of inputs x and x', we have that*

$$\prod_{i=j}^{k} A_{i,x_{inp(i)}} \neq \prod_{i=j}^{k} A_{i,x'_{inp(i)}}$$

The following claim is stated without proof, as again, it is very similar to the proof of Claim 5.2:

Claim 5.6. *Let $\mathcal{P}$ and $\mathcal{P}'$ be two strongly unique input equivalent branching programs. Then $\mathcal{P}$ and $\mathcal{P}'$ are indistinguishable via annihilation attacks.*

5.5 Closing thoughts.

Notice that in some sense we have managed to derive some "space" from the $A_{i,b}$ over which annihilating is impossible. In general, the $A_{i,b}$ may give rise to algebraic spaces which may give us a way to partition branching programs into sets of branching programs that are indistinguishable via annihilation attack (this is certainly an equivalence relationship). Whether these spaces can be easily characterized from the $A_{i,b}$ remains to be

seen. Additionally, pursuing the interpretation of branching programs as graphs may reveal a transformation under which partitioning branching programs can be interpreted as a graph isomorphism problem, in which case we have a wide range of both practical tools and theoretical literature to attack the problem.