Preventing CLT Attacks on Obfuscation with Linear Overhead

Rex Fernando* Peter M. R. Rasmussen* Amit Sahai*

March 17, 2017

Abstract

We describe a defense against zeroizing attacks on indistinguishability obfuscation (iO) over the CLT13 multilinear map construction that only causes an additive blowup in the size of the branching program. This defense even applies to the most recent extension of the attack by Coron *et al.* (ePrint 2016), under which a much larger class of branching programs is vulnerable. To accomplish this, we describe an attack model for the current attacks on iO over CLT13 by distilling an essential common component of all previous attacks.

This leads to the notion of a function being *input partionable*, meaning that the bits of the function's input can be partitioned into somewhat independent subsets. We find a way to thwart these attacks by requiring a "stamp" to be added to the input of every function. The stamp is a function of the original input and eliminates the possibility of finding the independent subsets of the input necessary for a zeroizing attack. We give three different constructions of such "stamping functions" and prove formally that they each prevent any input partition.

We also give details on how to instantiate one of the three functions efficiently in order to secure any branching program against this type of attack. The technique presented alters any branching program obfuscated over CLT13 to be secure against zeroizing attacks with only an additive blowup of the size of the branching program that is linear in the input size and security parameter.

We can also apply our defense to a recent extension of annihilation attacks by Chen *et al.* (ePrint 2016) on obfuscation over the GGH13 multilinear map construction.

*UCLA and Center for Encrypted Functionalities. {rex, rasmussen, sahai}@cs.ucla.edu. Research supported in part from a DARPA/ARL SAFEWARE award, NSF Frontier Award 1413955, NSF grants 1619348, 1228984, 1136174, and 1065276, a Xerox Faculty Research Award, a Google Faculty Research Award, an equipment grant from Intel, and an Okawa Foundation Research Grant. This material is based upon work supported by the Defense Advanced Research Projects Agency through the ARL under Contract W911NF-15-C-0205. The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense, the National Science Foundation, or the U.S. Government.

1 Introduction

Indistinguishability obfuscation (iO) has so far relied on multilinear maps for instantiation (e.g. [GGH$^+$13b]) and viable candidates for such are sparse. On top of that, the few that exist [GGH13a, CLT13, GGH15] have all been shown to suffer from significant vulnerabilities. However, not all attacks against these multilinear maps can be applied to iO. The very particular structure that most iO candidates induce puts numerous constraints on the way the encoded values can be combined, thus often not allowing the flexible treatment needed to mount an attack. Attacks on iO schemes have nonetheless been found for obfuscation of increasingly general families of functions.

In this paper we focus on the Coron-Lepoint-Tibouchi (CLT13) multilinear maps [CLT13]. The known attacks over CLT13 are called *zeroizing attacks* [CHL$^+$15, CGH$^+$15, CLLT17]. To be carried out, they require multiple zero encodings that are the result of multiplications of elements that satisfy a certain structure. Since obfuscations of matrix branching programs only produce zeroes when evaluated in a very specific manner, setting up such a zeroizing attack on an obfuscated branching program is rather non-trivial.

Because of this, the first paper applying zeroizing attacks over CLT13 to iO only showed how to apply the attack to very simple branching programs [CGH$^+$15], and attacking more realistic targets seemed out of reach of this technique. However a very recent work by Coron et al. [CLLT17] introduced a simple method that can transform a much larger class of branching programs into ones that have this very specific structure. As such, zeroizing attacks appear much more threatening to the security of iO over CLT13 than previously thought.

1.1 The Story so Far: Branching Programs and Zeroizing Attacks

This section will serve as a light introduction to the terminology and concepts at work in this paper.

Branching Programs. The "traditional" method of obfuscation works with matrix branching programs that encode boolean functions. A (single input) matrix branching program BP is specified by the following information. It has a length ℓ, input size n, input function $\textbf{inp}\colon [\ell] \to [n]$, square matrices $\{A_{i,b}\}_{i\in[\ell],b\in\{0,1\}}$, and bookend vectors A_0 and $A_{\ell+1}$. An evaluation of the branching program BP on input $x \in \{0,1\}^n$ is carried out by computing

$$A_0 \times \prod_{i=1}^{\ell} A_{i,x_{\textbf{inp}(i)}} \times A_{\ell+1}.$$

If the product is zero then $\mathsf{BP}(x) = 0$ and otherwise, $\mathsf{BP}(x) = 1$.

Multilinear Maps and Obfuscating Branching Programs. Current instantiations of iO are based on graded multilinear maps [GGH$^+$13b, BGK$^+$14]. This primitive allows values $\{a_i\}$ to be encoded to $\{[a_i]\}$ in such a manner that they are hidden. The multilinear map allows evaluation of a very restricted class of polynomials over these encoded values. Moreover, evaluating a polynomial, p, over the encodings in this way should only yield one bit of information: whether or not the result, $p(\{a_i\})$, is zero.

To obfuscate a branching program, we first randomize the matrices and then encode the entries of the matrix using a multilinear map. (See for example [BGK$^+$14] for details on how the matrices are randomized.) The hope is that the multilinear map will allow evaluations of the branching program but will not allow other malicious polynomials over the encodings that would violate indistinguishability. In fact, Barak *et al.* [BGK$^+$14] show that if zero testing the result of evaluations over the multilinear map truly only reveals whether or not the evaluation is zero and does not leak anything else then this scheme is provably secure.

Zeroizing Attacks on Obfuscated Branching Programs. Unfortunately, the assumption that zero-testing does not leak any information is unrealistic. In particular, the zeroizing attacks over the CLT13 multilinear map work by exploiting the information leaked during successful zero tests to obtain the secret parameters.

Before discussing the zeroizing attacks on iO, we first consider how the attacks work over raw encodings. (This version of the attack was first presented in [CHL$^+$15]) Each of the known zeroizing attacks require sets of encodings that satisfy a certain structure. Namely, to attack a CLT instance of dimension n, an adversary needs three sets of encodings $\{B_i\}_{i\in[n]}$, $\{C_0, C_1\}$, and $\{D_j\}_{j\in[n]}$ such that for every $i, j \in [n]$ and $\sigma \in \{0,1\}$, $B_i C_\sigma D_j$ is a top-level encoding of zero. In other words, we must be able to vary the choice of encoding in each set independently of the other choices and always produce an encoding of zero. If an adversary is able to obtain such sets, then the adversary is able to factor the modulus of the ciphertext ring, completely breaking the CLT instance. (We give more details about the attack in Section 2.)

In the case of applying zeroizing attacks to iO constructions over CLT13, the above requirement leads to an interesting constraint on the behavior of the function being obfuscated. Obfuscation schemes are designed so that the only way to achieve an encoding of zero is by performing an honest evaluation of the obfuscated program. In fact, [BGK$^+$14] prove that their obfuscation scheme has (a more formal version of) this property. The result of this is that the only way to obtain the products of the three sets of encodings described above is to vary the inputs to the obfuscated function. So for the obfuscation of a function to be vulnerable to zeroizing attacks over CLT13, the input bit indices of the function must be divisible into three sets, corresponding to the three sets of encodings above.

Input Partitioning. We describe a generalization of this condition on the input here. Let $f\colon \{0,1\}^n \to \{0,1\}$ be a function to be obfuscated. In order to deploy a zeroizing attack on the obfuscation of f over CLT13, it is required that the function allows an *input partitioning*. We say that there is an input partitioning of f if there exist sets $A \subseteq \{0,1\}^k, B \subseteq \{0,1\}^l, k+l = n$ and a permutation $\pi \in S_n$ such that $|A|, |B| > 1$ and for every $a \in A$ and $b \in B$, $f(\pi(a \mid\mid b)) = 0$, where π acts on the bit-string $a \mid\mid b$ by permuting its bits. In words, the function f can be input partitioned if the indices of the input can be partitioned into two sets such that varying the bits of the partitions independently within certain configurations will always yield zero as the output. As mentioned above, all known attacks on obfuscation over CLT13 require this partition to exist. In fact, the current attacks need a partition into three parts to succeed, but since any input partition into three parts can be viewed as an input partition into two parts, we treat that more general case instead.

It is worth noting that zeroizing attacks in fact require a stronger condition on the branching program in order to succeed; the matrices of the obfuscated branching program must be organized in a specific way in relation to the three sets of inputs. But preventing an input partition necessarily prevents this stronger condition. Since all known attacks require an input partition, this constitutes an "attack model" of sorts for obfuscation over CLT13. Previous authors [CGH$^+$15, CLLT17] have considered the stronger condition as a requirement for their attacks, but we are the first to consider the input partition of a boolean function as the basis of a formal attack model. We will define this model formally in Section 3.2.

1.2 Our Contributions

Our aim in this paper is to provide a robust defense against the known classes of zeroizing attacks for iO over CLT13 and against potential future extensions of these attacks. Further, we want the defense to have a minimal impact on the efficiency of the obfuscated program. In this section we describe how we achieve such a defense that only incurs an additive linear blowup.

More specifically, the goal of the paper is to construct a procedure which takes an input partitionable function $f\colon \{0,1\}^n \to \{0,1\}$ and produces a function g with the same functionality, but on which no input partitions exist. The existence of an input partition depends on which inputs cause g to output zero; note that a branching program is defined to output zero if the result of the multiplications of the matrices is zero and one if the result is any other value. So we will think of g as being a function $\{0,1\}^{n'} \to \{0,\perp\}$.

Input Stamping. The idea behind such a procedure, is to append a "stamp" to the end of the input of the function f. The stamp is designed to not allow the input as a whole to be input partitioned. More specifically, we will construct a function $h\colon \{0,1\}^n \to \{0,1\}^m$ such that given any $f\colon \{0,1\}^n \to \{0,1\}$ we can construct a new program $g\colon \{0,1\}^{n+m} \to \{0,\perp\}$ from f such that $g(s)$ outputs 0 if and only if the input is of the form $s = x \mid\mid h(x)$ and $f(x) = 0$ and otherwise outputs $\perp$. Note that the original $\{0,1\}$-output of f is recoverable from g as long as the evaluation took place with the correct stamp appended to the input.

Our main theoretical result is to find a necessary and sufficient condition on h such that g cannot be input partitioned. If this is the case then we say h *secures* f. We state a sufficient condition below as Theorem 1. In Section 3.4 we restate Theorem 1 with both necessary and sufficient conditions after introducing some preliminaries which are required for the stronger version of the theorem.

Theorem 1 (Weakened)**.** *Let $h\colon \{0,1\}^n \to \{0,1\}^m$. Let $x_{1,1}, x_{1,2}, x_{2,1}, x_{2,2} \in \{0,1\}^n$ be treated as integers. If whenever*

$$\begin{gathered} x_{1,1} - x_{1,2} = x_{2,1} - x_{2,2} \\ h(x_{1,1}) - h(x_{1,2}) = h(x_{2,1}) - h(x_{2,2}) \end{gathered}$$

it is the case that $x_{1,1} = x_{1,2}$ or $x_{1,1} = x_{2,1}$, then h secures all functions $f\colon \{0,1\}^n \to \{0,1\}$.

With this theorem, two questions arise: whether is is feasible to construct such an h, and how efficiently we can construct the modified g to be. The second question is relevant with respect to the work in [AGIS14, BISW17] on improving the efficiency of obfuscation of branching programs. It is especially relevant to [BISW17] since this paper uses CLT13 to achieve a significant speedup factor from previous constructions. Thus, establishing the security of obfuscation over CLT13 with minimal overhead is pertinent.

With regards to efficiency, the size of the image of h becomes important. Using an h that has an output size m which is large relative to n will necessarily affect the efficiency of the resulting g. In Section 3.5 we explore the minimum value of m necessary in order for h to be secure. We show that m must be at least linear in terms of n.

Constructions. The first two instantiations we present for h address the question of the feasibility of constructing such a function. They are both number-theoretic and follow from the fact that Theorem 1 can be interpreted as a sort of nonlinearity property. We show that squaring and exponentiation modulo a large enough prime satisfy this property and thus secure any function f. We stress that we do not rely on any number-theoretic assumptions in the proofs that these functions satisfy Theorem 1.

The third instantiation is a combinatorial function and is motivated by the desire for efficiency. To that end, instead of defining a single function which is guaranteed to have the property specified above, we define a family of very simple functions where the probability of a random choice from this family is very likely to have the property.

We define h as follows. Let k and t be parameters set beforehand. For each $i \in [t]$ and $j \in [n]$, choose $\pi_{i,j,0}$ and $\pi_{i,j,1}$ at random from the set of permutations acting on k elements. For an input $x \in \{0,1\}^n$, define

$$h_i(x) = (\pi_{i,1,x_1} \circ \pi_{i,2,x_2} \circ \cdots \circ \pi_{i,n,x_n})(1).$$

Then $h(x) = h_1(x) \mid\mid h_2(x) \mid\mid \cdots \mid\mid h_t(x)$.

We give a combinatorial probabilistic argument that with $k = O(1)$ and $t = O(n + \lambda)$ the choice of h secures all functions $f\colon \{0,1\}^n \to \{0,1\}$ with overwhelming probability as a function of λ.

Parallel Initialization. Since this construction for h is defined in terms of permutations and processes the input in the same way that a branching program does, constructing a branching program that computes such an h and subsequently modifying a branching program for f to create a branching program for the corresponding g is fairly straightforward. While this is already vastly more efficient than implementing the first two instantiations of h using a matrix branching program, running the functions h_i in sequence would cause a linear blowup in the size of the branching program. We do much better than this and achieve a constant blowup factor with the following trick. Unlike the GGH13 multilinear map construction, CLT13 allows for a composite ring size. We use this fact to evaluate all the h_i in parallel. This achieves a constant factor overhead.[1] (This technique was used, for example, in [AS17, GLSW15], albeit for different purposes.)

Perspectives. We remark that the attacks in [CLLT17] still do not apply to obfuscations of all branching programs. Specifically, if the branching program is too long compared to the input size then there is a blowup associated with the transformation in [CLLT17] which becomes infeasible. Also, it is not yet known how to apply the attack to dual-input branching programs, due to a similar blowup in complexity. Although this is the case, it is definitely possible that future work will extend zeroizing attacks to longer branching programs and dual-input branching programs. Our defense hedges against these possible future attacks, because it defends against any attack which requires an input partition.

It is noteworthy to contrast this line of work with the recent attacks on iO over the GGH13 [GGH13a] multilinear maps construction. In [MSZ16] Miles et al. implement the first known such attacks, which they call *annihilating attacks*. A follow-up paper [GMM+16] gives a weakened multilinear map model and an obfuscation construction in this model which is safe against all annihilating attacks. We stress that the attacks over CLT13 are not related to these annihilating attacks, which are not known to work over CLT13. However, a recent paper attacking obfuscation over GGH13 [CGH17], in which the authors extend annihilating attacks to the original GGHRSW construction of iO, does use an input partition as part of their attack. They do this as a first step in

[1] In fact our actual overhead is additive and linear in terms of the input size of f, not the size of its branching program. See Section 4.3 for details.

order order to recover a basis for the ideal which defines the plaintext space. Our defense applies to this step of their attack.

As a final note, our defense does not operate in a weak multilinear map model, in contrast to the one defined in [GMM$^+$16]. We leave it as an important open question to develop such a weak multilinear map model for CLT13.

Organization. In Section 2 we discuss the attacks on obfuscation over CLT13 in more detail. In Section 3 we define formally what it means for a function to be input partitionable, and give a necessary and sufficient condition for any h to secure a function. We also give our lower bound on the size of the image of h. Finally, in Section 4 we define and prove the correctness of our instantiations of h.

2 Recent attacks on CLT13

In this section we give a high-level overview of the new attack by Coron, Lee, Lepoint and Tibouchi [CLLT17]. We start by reviewing the older attacks in [CHL$^+$15, CGH$^+$15] which this attack is based on.

The basic idea behind all the zeroizing attacks over CLT13 is to exploit the specific structure of the zero-test of CLT13, which differs from the other multilinear map constructions. CLT13 works over a ring $\mathbb{Z}_{x_0} \cong \bigoplus_{i=1}^{k} \mathbb{Z}_{p_i}$, and zero-testing an element successfully results in a sum $\sum_{i=1}^{k} r_i \rho_i$ where r_i is the component of the element modulo p_i and ρ_i is specific to the particular CLT13 instantiation. All of the attacks follow the same basic outline: construct a matrix A where every $A_{i,j}$ is the result of a successful zero-test such that A is similar to another matrix that contains information about the secret parameters of the CLT13 instantiation. In order to do this, they start with three sets of encodings $\{B_i\}_{i\in[n]}$, $\{C^0, C^1\}$, and $\{D_j\}_{j\in[n]}$ such that for every $i, j \in [n]$ and $\sigma \in \{0,1\}$, $B_i C^\sigma D_j$ is a top-level encoding of zero. In the simplest version of the attack, putting these encodings into a matrix A' and taking $A = A'A'^{-1}$ results in a matrix that is similar to a diagonal matrix with eigenvalues c_i^0/c_i^1, where c_i^σ is the i-th component of C^σ. These eigenvalues can be used to factor x_0.

All known attacks on obfuscation over CLT13 have proceeded in a very similar manner to the method just described: since the evaluation of a branching program is a product of matrices over encodings in a multilinear map, they divide the steps of the branching program into three parts corresponding to the sets of encodings above such that these three parts can be varied independently of the others.

To see what we mean by this, let

$$M(x) = \widehat{M}_0 \times \prod_{i=1}^{r} \widehat{M}_{i,x_{\mathsf{inp}(i)}} \times \widehat{M}_{r+1}, x \in \{0,1\}^t$$

be an obfuscation of a matrix branching program. We try to find $B_x = \widehat{M}_0 \times \prod_{i=1}^{a} \widehat{M}_{i,x_{\mathsf{inp}(i)}}$, $C_x = \prod_{i=a+1}^{b} \widehat{M}_{i,x_{\mathsf{inp}(i)}}$, and $D_x = \prod_{i=b+1}^{r} \widehat{M}_{i,x_{\mathsf{inp}(i)}} \times M_{i,r+1}$ such that we can partition the input bits as $\mathcal{B} \uplus \mathcal{C} \uplus \mathcal{D} = [t]$ and the value of B_x, C_x, and D_x rely only on $\mathcal{B}$, $\mathcal{C}$, and $\mathcal{D}$, respectively. Write $M(bcd)$ to mean the evaluation of M where b specifies the bits with positions in $\mathcal{B}$, and with c, d likewise with $\mathcal{C}, \mathcal{D}$. We further try to find sets of bit strings $\mathscr{B}, \mathscr{C}, \mathscr{D}$ where $\mathscr{B}, \mathscr{D}$ are large and $\mathscr{C}$ is at least of size two and for all $b \in \mathscr{B}, c \in \mathscr{C}, d \in \mathscr{D}$, $M(bcd) = 0$. If we can do all this, then the corresponding products of matrices form products of zero which decompose in a similar manner to the products of encodings used for the previous attack, and can similarly be used to mount an attack on the CLT13 instance used.

The problem with using this attack directly is that only the very simplest of branching programs can be decomposed in this way. In particular, attacks on any branching program that makes several passes over its input are ruled out.

The modified attack in [CLLT17] overcomes this limitation with a matrix identity which allows a rearranging of the matrix product corresponding to a branching program execution. The identity is as follows:

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

Using this identity, [CLLT17] shows how to attack a branching program with input function $\mathsf{inp}(i) = \min(i, 2t+1-i)$ for $1 \leq i \leq 2t+1$. Note that any branching program with this function does not satisfy the property above which was required for the earlier CLT13 attacks, since every input bit except for the t-th bit controls two nonconsecutive positions in the branching program. We can write such a program evalution as

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

where $B(x)$ and $B'(x)$ are both controlled by the same inputs, and likewise for $C(x)$ and $C'(x)$. [CLLT17] show that this can be rewritten as

$$(B'(x)^T \otimes B(x)) \times (C'(x)^T \otimes C(x)) \times \mathsf{vec}(D(x)) \times p_{zt} \pmod{x_0}$$

where now the three sets of inputs control consecutive pieces of the product. They then show how to use a modification of the original attack on this product, factoring x_0.

3 Securing Functions against Partition Attacks

3.1 Notation

We first introduce some notation for our exposition.

Definition 1. *For any positive integer $k \in \mathbb{N}$ we denote by $\mathbb{Z}_k$ the set $\mathbb{Z}/k\mathbb{Z}$.*

Definition 2. *Let $n \in \mathbb{N}$ be a positive integer and $\vec{v} \in \mathbb{N}^n$ a vector. We will denote by $\mathbb{Z}_{\vec{v}}$ the set*

$$\mathbb{Z}_{v_1} \times \mathbb{Z}_{v_2} \times \cdots \times \mathbb{Z}_{v_n}.$$

In this and following sections we will consider functions f that we want to secure and input stamping functions h. We will consider such functions as having domains and/or codomains of the form $\mathbb{Z}_{\vec{v}}$. Note that if we define $\vec{v} = (2, 2, \ldots, 2)$, then $\mathbb{Z}_{\vec{v}} = \{0,1\}^n$, so this is a a generalization of binary functions. We do this because in the instantiations section we will define an h which needs this generalized input format. Thus we state all theorems using this more general format to accommodate such instantiations.

Definition 3. *For a positive integer $n \in \mathbb{N}$ we denote by $[n]$ the set $\{1, 2, \ldots, n\}$.*

Definition 4. *For a positive integer $t \in \mathbb{N}$, we denote by S_t the set of permutations of the set $[t]$.*

Definition 5. *For two vectors or strings a and b let $a \,||\, b$ denote their concatenation.*

3.2 Input Partition Attacks

In this section we define formally the notion of an input partition attack. We also define what it means for a function to be hard or impossible to partition. Since in this section we only are concerned with whether a function outputs zero or not, we consider functions with codomain $\{0, \perp\}$, where $\perp$ represents any nonzero branching program output.

Definition 6 (Input Partition)**.** *Let $\vec{v} \in \mathbb{N}^t$ be a vector and $f \colon \mathbb{Z}_{\vec{v}} \to \{0, \perp\}$ be a function. An* input partition *for f of degree k is a tuple*

$$\mathcal{I}_f^k = \left(\sigma \in S_t,\ \{a_i\}_{i \in [k]} \subseteq \mathbb{Z}_{\vec{u}_1},\ \{c_j\}_{j \in [k]} \subseteq \mathbb{Z}_{\vec{u}_2}\right)$$

satisfying $a_i \neq a_j$ and $c_i \neq c_j$ for all $i, j \in [k]$ with $i \neq j$ and $\sigma(\vec{u}_1 \mid\mid \vec{u}_2) = \vec{v}$ such that for all $i, j \in [k]$,

$$f(\sigma(a_i c_j)) = 0.$$

Definition 7 (Input Partition Attack)**.** *For each $t \in \mathbb{N}$ let $\vec{v}_t \in \mathbb{N}^t$ be a vector, $\mathcal{F}_t$ be a family of functions $f \colon \mathbb{Z}_{\vec{v}_t} \to \{0, \perp\}$, and let $\mathcal{F} = \{\mathcal{F}_t\}_{t \in \mathbb{N}}$. We say that a PPT adversary $\mathcal{A}$ performs an input partition attack of degree k on $\mathcal{F}$ if for a non-negligable function ϵ,*

$$\Pr_{w, f \leftarrow \mathcal{F}_t}\left[\mathcal{A}(f) = \mathcal{I}_f^k \text{ is an input partition of } f \text{ of degree } k\right] > \epsilon(t),$$

where the probability is taken over the randomness w of $\mathcal{A}$ and the choice of f.

Turning the above definition around, we can ensure security against input partition attacks if the function we obfuscate satisfies the following.

Definition 8 (Input Partition Resistance)**.** *For each $t \in \mathbb{N}$ let $\vec{v}_t \in \mathbb{N}^t$ be a vector, $\mathcal{F}_t$ be a family of functions $f \colon \mathbb{Z}_{\vec{v}_t} \to \{0, \perp\}$, and let $\mathcal{F} = \{\mathcal{F}_t\}_{t \in \mathbb{N}}$. We say that $\mathcal{F}$ is* input partition resistant *for degree k if no PPT adversary successfully performs an independent input attack on f of degree k.*

A stronger version of this is for a function to simply not admit any input partitions which would clearly make attacks requiring a partition of the input impossible.

Definition 9 (Input Unpartitionable Function)**.** *Let $\vec{v} \in \mathbb{N}^t$ be a vector and $f \colon \mathbb{Z}_{\vec{v}} \to \{0, \perp\}$ be a function. We say that f is* input unpartitionable *for degree k if it does not admit an input partition of degree k. If f is input unpartitionable for degree 2, we simply say that it is* input unpartitionable*.*

3.3 Securing Functions

Now that we have defined the type of attack we aim to defend against, we introduce the input "stamping" function h and define what it means for h to secure a function f.

Definition 10 (Securing a Function)**.** *Let $\vec{v}_1 \in \mathbb{N}^{t_1}, \vec{v}_2 \in \mathbb{N}^{t_2}$ be vectors and write $\vec{v} = \vec{v}_1 \mid\mid \vec{v}_2$. Let $f \colon \mathbb{Z}_{\vec{v}_1} \to \{0, 1\}$ and $h \colon \mathbb{Z}_{\vec{v}_1} \to \mathbb{Z}_{\vec{v}_2}$ be functions and construct a function $g \colon \mathbb{Z}_{\vec{v}} \to \{0, \perp\}$ as follows:*

$$g(ab) = \begin{cases} f(a), & h(a) = b \\ \perp, & h(a) \neq b. \end{cases}$$

We say that h completely secures *f if g is input unpartitionable.*

A slightly less strict definition is the following which defines what it means for a function family to statistically secure a function.

Definition 11 (Statistically Securing a Function)**.** *Let $\vec{v} \in \mathbb{N}^t$ be a vector, f be a function $f\colon \mathbb{Z}_{\vec{v}} \to \{0,1\}$, and $\mathcal{H}$ be a collection $\mathcal{H} = \{\mathcal{H}_\lambda\}_{\lambda\in\mathbb{N}}$ of function families such that $\mathcal{H}_\lambda$ is a family of functions $h\colon \mathbb{Z}_{\vec{v}} \to \mathbb{Z}_{\vec{u}_\lambda}$ for some $\vec{u}_\lambda \in \mathbb{N}^{k_\lambda}$. We say that $\mathcal{H}$* statistically secures *f if for some negligible function ϵ and for all $\lambda \in \mathbb{N}$,*

$$\Pr_{h \xleftarrow{\$} \mathcal{H}_\lambda}[h \textit{ completely secures } f] > 1 - \epsilon(\lambda),$$

where h is sampled uniformly from $\mathcal{H}_\lambda$.

3.4 Necessary and Sufficient Conditions

In this section we present and prove the necessary and sufficient condition on a function h in order for it to secure every function f. First we give some useful definitions.

Definition 12. *Let $k \in \mathbb{N}$ be a positive integer and define the equivalence relation $\sim$ on $\mathbb{Z}_k \times \mathbb{Z}_k$ as follows. Two elements $(a,b),(c,d) \in \mathbb{Z}_k \times \mathbb{Z}_k$ are equivalent under $\sim$ if and only if either $(a,b) = (c,d)$ or $a = b$ and $c = d$. We denote by $\mathcal{Z}_k$ the set*

$$\mathcal{Z}_k = \mathbb{Z}_k \times \mathbb{Z}_k / \sim .$$

For a vector $\vec{v} \in \mathbb{N}^t$ we write $\mathcal{Z}_{\vec{v}}$ for the set $\mathcal{Z}_{v_1} \times \mathcal{Z}_{v_2} \times \cdots \times \mathcal{Z}_{v_t}$.

Definition 13. *Let $\vec{v} \in \mathbb{N}^t$ be a vector. Define an operation $*\colon \mathbb{Z}_{\vec{v}} \times \mathbb{Z}_{\vec{v}} \to \mathcal{Z}_{\vec{v}}$ as follows. For two elements $(a_1,\dots,a_t),(b_1,\dots,b_t) \in \mathbb{Z}_{\vec{v}}$,*

$$(a_1,\dots,a_t) * (b_1,\dots,b_t) = ((a_1,b_1),\dots,(a_t,b_t)) \in \mathcal{Z}_{\vec{v}}.$$

The operation $*$ is essentially a projection of two vectors $\vec{a},\vec{b} \in \mathbb{Z}_{\vec{v}}$ into $\mathcal{Z}_{\vec{v}}$.

We now give the characterization.

Definition 14 (Safe Function)**.** *Let $\vec{v}_1 \in \mathbb{N}^{t_1}, \vec{v}_2 \in \mathbb{N}^{t_2}$ be vectors. A function $h\colon \mathbb{Z}_{\vec{v}_1} \to \mathbb{Z}_{\vec{v}_2}$ is* safe *if for every $x_{1,1}, x_{1,2}, x_{2,1}, x_{2,2} \in \mathbb{Z}_{v_1}$ it is the case that if both of the following hold:*

$$x_{1,1} * x_{1,2} = x_{2,1} * x_{2,2}$$
$$h(x_{1,1}) * h(x_{1,2}) = h(x_{2,1}) * h(x_{2,2}),$$

then $x_{1,1} = x_{1,2}$ or $x_{1,1} = x_{2,1}$.

Theorem 1. *Let $\vec{v}_1 \in \mathbb{N}^{t_1}, \vec{v}_2 \in \mathbb{N}^{t_2}$ be vectors. The function $h\colon \mathbb{Z}_{\vec{v}_1} \to \mathbb{Z}_{\vec{v}_2}$ completely secures every function $f\colon \mathbb{Z}_{\vec{v}_1} \to \{0,1\}$ if and only if it is safe.*

In order to prove Theorem 1 we first state and prove two lemmas.

Lemma 1. *Let $\vec{v}_1 \in \mathbb{N}^{t_1}$ and $\vec{v}_2 \in \mathbb{N}^{t_2}$ be vectors and $\sigma \in S_{t_1+t_2}$. Let $a_1, a_2 \in \mathbb{Z}_{\vec{v}_1}$, $c_1, c_2 \in \mathbb{Z}_{\vec{v}_2}$, and*

$$\{r_1,\dots,r_k\} = T \subseteq [t_1+t_2], r_1 < r_2 < \cdots < r_k.$$

Finally, define a function p_T such that for $x \in \mathbb{Z}_{\vec{v}_1||\vec{v}_2}$, $p_T(x) = x_{r_1} x_{r_2} \dots x_{r_k}$. Then

$$p_T(\sigma(a_1c_1)) * p_T(\sigma(a_2c_1)) = p_T(\sigma(a_1c_2)) * p_T(\sigma(a_2c_2))$$

Proof. First, note that this lemma holds in general if and only if it holds for $T = [t_1 + t_2]$. So we will simply show that

$$\sigma(a_1c_1) * \sigma(a_2c_1) = \sigma(a_1c_2) * \sigma(a_2c_2)$$

which is equivalent to showing that

$$a_1c_1 * a_2c_1 = a_1c_2 * a_2c_2.$$

However, this is trivial from the definition of the operation $*$ and the conclusion follows. $\square$

Lemma 2. *Let $\vec{v} \in \mathbb{N}^t$ be a vector and let $x_{1,1}, x_{1,2}, x_{2,1}, x_{2,2} \in \mathbb{Z}_{\vec{v}}$ be given satisfying $x_{1,1} \neq x_{1,2}$ and $x_{1,1} \neq x_{2,1}$ and*

$$x_{1,1} * x_{1,2} = x_{2,1} * x_{2,2}.$$

Then there exist

$$\sigma \in S_t,\ a_1, a_2 \in \mathbb{Z}_{\vec{v}_1}, c_1, c_2 \in \mathbb{Z}_{\vec{v}_2}$$

with $a_1 \neq a_2$, $c_1 \neq c_2$, and $\sigma(\vec{v}_1 \mid\mid \vec{v}_2) = \vec{v}$ such that for every $i, j \in \{1, 2\}$,

$$\sigma(a_ib_j) = x_{i,j}$$

Proof. Let S be the set of indices $j \in [t]$ such that $x^j_{1,1} = x^j_{1,2}$ and let D be the set of indices $j \in [t]$ such that $x^j_{1,1} \neq x^j_{1,2}$. It is clear that S and D partition the set of indices $[t]$. Since $x_{1,1} * x_{1,2} = x_{2,1} * x_{2,2}$, we must have the following relations:

$$\forall j \in S\colon\ x^j_{1,1} = x^j_{1,2} \text{ and } x^j_{2,1} = x^j_{2,2} \tag{1}$$

$$\forall j \in D\colon\ x^j_{1,1} = x^j_{2,1} \neq x^j_{1,2} = x^j_{2,2} \tag{2}$$

Note that because $x_{1,1} \neq x_{2,1}$, S and D must be non-empty and there must exist an index $r \in S$ such that $x^r_{1,1} = x^r_{1,2} \neq x^r_{2,1} = x^r_{2,2}$.

Now, enumerating S and D as $S = \{m_1, \ldots, m_k\}$ and $D = \{n_1, \ldots, n_l\}$ with $k + l = t$, we set

$$a_i = x^{m_1}_{i,1} x^{m_2}_{i,1} \ldots x^{m_k}_{i,1} = x^{m_1}_{i,2} x^{m_2}_{i,2} \ldots x^{m_k}_{i,2}$$
$$c_j = x^{n_1}_{1,j} x^{n_2}_{1,j} \ldots x^{n_k}_{1,j} = x^{n_1}_{2,j} x^{n_2}_{2,j} \ldots x^{n_k}_{2,j},$$

for $i, j \in \{1, 2\}$ where the equalities to the right follow from (1) and (2), $a_1 \neq a_2$ because of the existence of r as above, and $c_1 \neq c_2$ from the definition of D.

Letting $\sigma \in S_t$ be the permutation such that

$$\sigma(m_1\, m_2\, \ldots\, m_k\, n_1\, n_2\, \ldots\, n_l) = 1\, 2\, \ldots\, t,$$

we find that $\sigma(a_ic_j) = x_{i,j}$ for every $i, j \in \{1, 2\}$ and we are done.

$\square$

Proof of Theorem 1. First, suppose that h completely secures every function f and assume for contradiction that there exists $x_{1,1}, x_{1,2}, x_{2,1}, x_{2,2} \in \mathbb{Z}_{\vec{v}_1}$ with $x_{1,1} \neq x_{1,2}$ and $x_{1,1} \neq x_{2,1}$ such that

$$x_{1,1} * x_{1,2} = x_{2,1} * x_{2,2}$$
$$h(x_{1,1}) * h(x_{1,2}) = h(x_{2,1}) * h(x_{2,2}).$$

Let f be the function satisfying $f(x) = 0$ for every $x \in \mathbb{Z}_{\vec{v}_1}$ and consider the function

$$g(ab) = \begin{cases} f(a), & h(a) = b \\ \perp, & h(a) \neq b. \end{cases}$$

We clearly have

$$(x_{1,1} \mid\mid h(x_{1,1})) * (x_{1,2} \mid\mid h(x_{1,2})) = (x_{2,1} \mid\mid h(x_{2,1})) * (x_{2,2} \mid\mid h(x_{2,2}))$$

and thus, by Lemma 2 there exist

$$\sigma \in S_{t_1+t_2},\ a_1, a_2 \in \mathbb{Z}_{\vec{u}_1}\ c_1, c_2 \in \mathbb{Z}_{\vec{u}_2}$$

with $a_1 \neq a_2$, $c_1 \neq c_2$, and $\sigma(\vec{u}_1 \mid\mid \vec{u}_2) = \vec{v}_1 \mid\mid \vec{v}_2$ such that for every $i, j \in \{1, 2\}$,

$$\sigma(a_i c_j) = x_{i,j} h(x_{i,j}).$$

However, then $g(\sigma(a_i c_j)) = 0$ for every $i, j \in \{1, 2\}$ which is a contradiction since g would be input unpartitionable if h completely secured the function f.

Second, suppose that h is safe and let $f \colon \mathbb{Z}_{\vec{v}_1} \to \{0, 1\}$ be arbitrary. Define

$$g(ab) = \begin{cases} f(a), & h(a) = b \\ \perp, & h(a) \neq b. \end{cases}$$

and assume for contradiction that there exists an input partition for g of degree two,

$$\mathcal{I}_g^2 = \left(\sigma \in S_{t_1+t_2},\ \{a_i\}_{i \in [k]} \subseteq \mathbb{Z}_{\vec{u}_1},\ \{c_j\}_{j \in [k]} \subseteq \mathbb{Z}_{\vec{u}_2}\right).$$

For each $i, j \in \{1, 2\}$, write $\sigma(a_i c_j) = x_{i,j} y_{i,j}$ with $x_{i,j} \in \mathbb{Z}_{\vec{v}_1}$ and $y_{i,j} \in \mathbb{Z}_{v_2}$ and observe that then $h(x_{i,j}) = y_{i,j}$. Furthermore, we have

$$x_{1,1} * x_{1,2} = x_{2,1} * x_{2,2}$$
$$y_{1,1} * y_{1,2} = y_{2,1} * y_{2,2}.$$

by Lemma 1. Since $h(x_{i,j}) = y_{i,j}$ it follows directly from the condition on h that either $x_{1,1} = x_{1,2}$ or $x_{1,1} = x_{2,1}$. The two cases are symmetric, so assume without loss of generality that $x_{1,1} = x_{1,2}$. Then $y_{1,1} = y_{1,2}$ and we get $\sigma(a_1 c_1) = \sigma(a_1 c_2)$. A contradiction as we required $c_1 \neq c_2$.

□

3.5 Lower Bound on the Output Size of Safe Functions

An implementation of a safe function h to secure a functions f by constructing the function g of Definition 10 results in an increase in the input size of f and further this extra input must be checked against the output of h. In the context of matrix branching programs, the check of the extra input requires a pass over the input, adding more matrices, which when initialized over multilinear maps is rather costly. Thus, knowledge about the minimal output size of a safe function is helpful in determining the costs of securing a function. In this section we show that this output size is at least linear in the input size of f.

Theorem 2. *Let $\vec{v}_1 = (v_1^1, v_1^2, \ldots, v_1^{t_1}) \in \mathbb{N}^{t_1}$ and $\vec{v}_2 = (v_2^1, v_2^2, \ldots, v_2^{t_2}) \in \mathbb{N}^{t_2}$ be vectors and let $h\colon \mathbb{Z}_{\vec{v}_1} \to \mathbb{Z}_{\vec{v}_2}$ be a safe function. If $v_1^k = \min_{1\leq i\leq t_1}(v_1^i)$ then*

$$\prod_{\substack{1\leq i\leq t_1\\ i\neq k}}^{t_1} v_1^i \leq \prod_{i=1}^{t_2}(v_2^i(v_2^i-1)+1).$$

Proof. Assume without loss of generality that $k=1$. Recall that the elements of $\mathcal{Z}_{\vec{v}_1^i}$ are $(a,b), a\neq b$ for $a,b \in \mathbb{Z}_{v_1^i}$ together with the single element consisting of the $\sim$-equivalence class $\{(b,b) \mid b \in \mathbb{Z}_{\vec{v}_1^i}\}$ that we denote by (a,a). Now, consider the vector

$$y = ((b,c),(a,a),(a,a),\ldots,(a,a)) \in \mathcal{Z}_{\vec{v}_1}$$

and let $T = \{(x_1,x_2) \in (\mathbb{Z}_{\vec{v}_1})^2 \mid x_1 * x_2 = y\}$. We have $|T| = \prod_{i=2}^{t_1} v_i$ since for the first index there is only the choice $x_1^1 = b$ and $x_2^2 = c$ and for index $i>1$ there are v_1^i choices for $x_1^i = x_2^i$. Now, define the function $t\colon T \to \mathcal{Z}_{\vec{v}_2}$ by $t(x_1,x_2) = h(x_1) * h(x_2)$. By the definition of a safe function, t must be injective. It follows that

$$\prod_{i=2}^{t_1} v_i = |T| \leq |\mathcal{Z}_{\vec{v}_2}| = \prod_{i=1}^{t_2}(v_2^i(v_2^i-1)+1).$$

□

Corollary 1. *Let $h\colon \{0,1\}^{t_1} \to [k]^{t_2}$ be a safe function. Then*

$$\frac{t_1-1}{\log(k(k-1)+1)} \leq t_2.$$

Proof. By Theorem 2, we have

$$2^{t_1-1} \leq (k(k-1)+1)^{t_2}.$$

Taking the logarithm on both sides of the equation yields the conclusion.

□

We will see in the instantiations of safe functions that while we do not achieve an optimal construction, all our constructions have output size linear in the input size of the original function and with fairly small coefficients, so they are asymptotically optimal.

4 Instantiations

We now present three instantations for h. We first give two number theoretic functions which secure any function f, and then we give a combinatorial function that statistically secures any function f.

4.1 Number Theoretical Functions

By the necessary and sufficient condition of Theorem 1 and the definition of a safe function, it seems that a function will secure every other function if it is somewhat non-linear everywhere. This is captured in the following corollary, letting us work with functions over the integers.

Corollary 2. *Let $h\colon \{0,1\}^n \to \{0,1\}^m$ be a function satisfying that for all $x_{1,1}, x_{1,2}, x_{2,1}, x_{2,2} \in \{0,1\}^n$ satisfying the equations*

$$x_{1,1} - x_{1,2} = x_{2,1} - x_{2,2}$$
$$h(x_{1,1}) - h(x_{1,2}) = h(x_{2,1}) - h(x_{2,2}).$$

when viewed as positive integers either $x_{1,1} = x_{1,2}$ or $x_{1,1} = x_{2,1}$. Then h completely secures every function $f\colon \{0,1\}^n \to \{0,\perp\}$.

Proof. This follows immediately from Theorem 1 since $x_{1,1} * x_{1,2} = x_{2,1} * x_{2,2}$ implies $x_{1,1} - x_{1,2} = x_{2,1} - x_{2,2}$ as integers.

□

Intuitively, many functions we know and love would satisfy this as long as they have sufficient non-linearity. Here we list two examples. In terms of output size, note that these functions both produce $n+1$ bits of output, where the minimum possible by Corollary 1 is $\frac{n-1}{\log 3}$. So the output size of these two functions is close to optimal.

Proposition 1. *Let p be a prime satisfying $2^n < p < 2^{n+1}$. The function $h\colon \{0,1\}^n \to \{0,1\}^{n+1}$ given by $h(x) = [x^2]_p$ completely secures every function $f\colon \{0,1\}^n \to \{0,\perp\}$.*

Proof. Let $x_{1,1}, x_{x_1,2}, x_{2,1}, x_{2,2} \in \{0,1\}^n$ be given satisfying $x_{1,1} - x_{2,1} = x_{1,2} - x_{2,2}$ and $h(x_{1,1}) - h(x_{2,1}) = h(x_{1,2}) - h(x_{2,2})$. We will show that $x_{1,1} = x_{1,2}$ or $x_{1,1} = x_{2,1}$, concluding the proof by Corollary 2.

Directly from the conditions on the $x_{i,j}$, we get

$$\begin{aligned}(x_{1,1} + x_{2,1})(x_{1,1} - x_{2,1}) &\equiv h(x_{1,1}) - h(x_{2,1}) \\ &= h(x_{1,2}) - h(x_{2,2}) \\ &\equiv (x_{1,2} + x_{2,2})(x_{1,2} - x_{2,2}) \pmod{p}.\end{aligned}$$

This yields two cases. If $x_{1,1} - x_{2,1} = x_{1,2} - x_{2,2} = 0$ then $x_{1,1} = x_{2,1}$ and we are done. Otherwise $x_{1,1} - x_{2,1} = x_{1,2} - x_{2,2}$ is invertible modulo p since $p > 2^n$ and we get

$$x_{1,1} + x_{2,1} \equiv x_{1,2} + x_{2,2} \pmod{p}.$$

Adding $x_{1,1} - x_{2,1} = x_{1,2} - x_{2,2}$ to both sides yields $2x_{1,1} \equiv 2x_{1,2} \pmod{p}$ which is equivalent to $x_{1,1} \equiv x_{1,2} \pmod{p}$. Hence, $x_{1,1} = x_{1,2}$ since $p > 2^n$ and we are done.

□

Proposition 2. *Let p be a prime satisfying $2^n < p < 2^{n+1}$ with primitive root r. The function $h\colon \{0,1\}^n \to \{0,1\}^{n+1}$ given by $h(x) = [r^x]_p$ completely secures every function $f\colon \{0,1\}^n \to \{0,1\}$.*

Proof. Let $x_{1,1}, x_{x_1,2}, x_{2,1}, x_{2,2} \in \{0,1\}^n$ be given satisfying $x_{1,1} - x_{2,1} = x_{1,2} - x_{2,2}$ and $h(x_{1,1}) - h(x_{2,1}) = h(x_{1,2}) - h(x_{2,2})$. We will show that $x_{1,1} = x_{1,2}$ or $x_{1,1} = x_{2,1}$, concluding the proof by Corollary 2.

Directly from the conditions on the $x_{i,j}$, we get

$$\begin{aligned} r^{x_{1,1}}(1 - r^{x_{2,1}-x_{1,1}}) &\equiv h(x_{1,1}) - h(x_{2,1}) \\ &= h(x_{1,2}) - h(x_{2,2}) \\ &\equiv r^{x_{1,2}}(1 - r^{x_{2,2}-x_{1,2}}) \pmod p. \end{aligned}$$

Now we have two cases. First, if $1 - r^{x_{2,1}-x_{1,1}} = 1 - r^{x_{2,2}-x_{1,2}}$ is invertible modulo p then $r^{x_{1,1}} \equiv r^{x_{1,2}} \pmod p$, yielding $x_{1,1} = x_{1,2}$ since r has order $p - 1 \geq 2^n$ modulo p. Second, if $1 - r^{x_{2,1}-x_{1,1}} = 1 - r^{x_{2,2}-x_{1,2}}$ is not invertible modulo p then clearly $1 - r^{x_{2,1}-x_{1,1}} \equiv 0 \pmod p$ and thus, $x_{2,1} - x_{1,1} = 0$ since the order of r is $\geq 2^n$. It follows that either $x_{1,1} = x_{1,2}$ or $x_{1,1} = x_{2,1}$. □

4.2 Permutation Hash Functions

We now discuss an instantiation which statistically secures functions f, as opposed to the functions in the previous section which completely secure f. Number theoretical functions like the ones in the previous section are difficult and expensive to implement in the setting of matrix branching programs since these do not generally support operations over a fixed field $\mathbb{Z}_p$. However, matrix branching programs naturally implement operations on the group of permutations, S_k. The function we define in this section are defined in terms of randomly chosen permutations, and turns out to be a much more practical alternative.

Definition 15. *A k-*Permutation Hash Function *of input size n is a function $h\colon \{0,1\}^n \to [k]$ randomly drawn as follows as follows. Select permutations $\{\pi_{i,b}\}_{i\in[n],b\in\{0,1\}} \xleftarrow{\$} S_k^{2n}$ uniformly at random. For an input $x \in \{0,1\}^n$ let*

$$\sigma_x = \prod_{i=1}^{n} \pi_{i,x_i}.$$

Then $h(x) = \sigma_x(1)$.

Lemma 3. *Let $x_{1,1}, x_{1,2}, x_{2,1}, x_{2,2} \in \{0,1\}^n$ be given such that*

$$x_{1,1} * x_{1,2} = x_{2,1} * x_{2,2},$$

$x_{1,1} \neq x_{1,2}$, and $x_{1,1} \neq x_{2,1}$. Then

$$\Pr_{h \xleftarrow{\$} S_k^{2n}} [h(x_{1,1}) * h(x_{1,2}) = h(x_{2,1}) * h(x_{2,2})] \leq \frac{k}{(k-1)^2}.$$

Proof. Write $u = x_{1,1} * x_{1,2} = x_{2,1} * x_{2,2}$ and denote by $x_{a,b}^i$ the ith bit of $x_{b,c}$ and by u_i the ith entry of u. Let $S_d \subset [n], d \in \mathcal{Z}_2$ be the set of indices $i \in [n]$ such that $u_i = d$, where we recall that the elements of $\mathcal{Z}_2$ are the equivalence classes $(0,1),(1,0),(0,0) \sim (1,1)$. We will denote the equivalence class containing $(0,0)$ and $(1,1)$ by (a,a). Thus, the set of indices $[n]$ is partitioned into $[n] = S_{(0,1)} \cup S_{(1,0)} \cup S_{(a,a)}$.

Now, it must be the case that there is a $j \in S_{(a,a)}$ such that $x_{1,1}^j = x_{1,2}^j \neq x_{2,1}^j = x_{2,2}^j$. To see this, note that $x_{1,1}$ and $x_{2,1}$ are identical at the indices of $S_{(0,1)}$ and $S_{(1,0)}$ and if $x_{1,1}$ and $x_{2,1}$ also are identical at the indices of $S_(a,a)$ then $x_{1,1} = x_{2,1}$ contrary to assumption. Choose j to be maximal. We can assume without loss of generality that there is an $l \in S_{(0,1)} \cup S_{(1,0)}$ such that $l > j$ as follows. Suppose that this is not the case. Then $x_{1,1}^i = x_{1,2}^i = x_{2,1}^i = x_{2,2}^i$ for every $i > j$ since j was maximal and we must have $i \in S_{(a,a)}$ whenever $i > j$. Now, consider the equation $u' = x_{1,1} * x_{2,1} = x_{1,2} * x_{2,2}$ where we simply swap $x_{1,2}$ and $x_{2,1}$ from our original expression. Let $S'_d \subset [n], d \in \mathcal{Z}_2$ be the set of indices $i \in [n]$ such that $u'_i = d$. In this dual situation, $j \in [n] \setminus S'_{(a,a)}$ and still $x_{1,1}^i = x_{1,2}^i = x_{2,1}^i = x_{2,2}^i$ for every $i > j$. Further, we can find a new maximal $j' \in S'_{(a,a)}$ with $j' < j$ such that $x_{1,1}^{j'} \neq x_{1,2}^{j'}$. Thus, in the dual situation j is the l that we were seeking in the original case. From this it follows that we without loss of generality can choose $l > j$ as above. Define the following permutations, noting that $x_{1,1}^i = x_{2,1}^i$ and $x_{1,2}^i = x_{2,2}^i$ for $i > j$.

$$\tau = \prod_{i=j+1}^{n} \pi_{i,x_{1,1}^i}$$
$$\sigma = \prod_{i=j+1}^{n} \pi_{i,x_{1,2}^i}$$
$$\gamma_{b,c} = \prod_{i=1}^{j-1} \pi_{i,x_{b,c}^i}, b,c \in \{1,2\}.$$

Then letting $b = x_{1,1}^j = x_{1,2}^j$ we get

$$h(x_{1,1}) = \gamma_{1,1} \circ \pi_{j,b} \circ \tau(1), \;\; h(x_{2,1}) = \gamma_{2,1} \circ \pi_{j,1-b} \circ \tau(1)$$
$$h(x_{1,2}) = \gamma_{1,2} \circ \pi_{j,b} \circ \sigma(1), \;\; h(x_{2,2}) = \gamma_{2,2} \circ \pi_{j,1-b} \circ \sigma(1).$$

Now, since $x_{1,1}^l = x_{2,1}^l \neq x_{1,2}^l = x_{2,2}^l$, the permutations $\pi_{l,x_{1,1}^l} = \pi_{l,x_{2,1}^l}$ and $\pi_{l,x_{1,2}^l} = \pi_{l,x_{2,2}^l}$ are chosen independently of each other. Thus, it follows from $l > j$ that over the choice of $\{\pi_{i,b}\}$ the permutation τ and the permutation σ are independently and uniformly distributed. Writing $(a,b) = (\tau(1), \sigma(1)) \in \mathbb{Z}_k \times \mathbb{Z}_k$, this means that (a,b) will be uniformly distributed in $\mathbb{Z}_k \times \mathbb{Z}_k$ over the choice of $\{\pi_{i,b}\}$. Thus, with probability $\frac{k-1}{k}$ we have $a \neq b$. We will assume that from now on. The above equations now become

$$h(x_{1,1}) = \gamma_{1,1} \circ \pi_{i,b}(a), \;\; h(x_{2,1}) = \gamma_{2,1} \circ \pi_{i,1-b}(a)$$
$$(x_{1,2}) = \gamma_{1,2} \circ \pi_{i,b}(b), \;\; h(x_{2,2}) = \gamma_{2,2} \circ \pi_{i,1-b}(b).$$

Noting that $\pi_{i,b}$ and $\pi_{i,1-b}$ are chosen independently we get that for some $(q,r),(s,t) \in T$ for $T = \mathbb{Z}_k \times \mathbb{Z}_k \setminus \{(a,a) \mid a \in \mathbb{Z}_k\}$ which are independent of each other and each uniformly distributed over T over the choice of $\{\pi_{i,b}\}$, we have

$$h(x_{1,1}) = \gamma_{1,1}(q), \;\; h(x_{2,1}) = \gamma_{2,1}(s)$$
$$(x_{1,2}) = \gamma_{1,2}(r), \;\; h(x_{2,2}) = \gamma_{2,2}(t).$$

The γs are chosen independently of the other permutations, so even conditional on the particular choice of the γs the distribution of (q, r) and (s, t) are independent and uniformly distributed across T. This means that the pairs $(h(x_{1,1}), h(x_{1,2}))$ and $(h(x_{2,1}, h(x_{2,2}))$ are independent of each other and are uniformly distributed over subsets $U_1, U_2 \subset [k] \times [k]$, respectively, of size $|T| = k(k-1)$.

Now, taking into account the case we disregarded in the beginning where $a = b$ which has probability $\frac{1}{k}$, we can write

$$\begin{aligned}
\Pr_{h \xleftarrow{\$} S_k^{2n}} [h(x_{1,1}) * h(x_{1,2}) = h(x_{2,1}) * h(x_{2,2})] &\leq \frac{1}{k} + \Pr_{\substack{(\alpha,\beta) \xleftarrow{\$} U_1 \\ (\gamma,\delta) \xleftarrow{\$} U_2}} [(\alpha, \beta) \sim (\gamma, \delta)] \\
&= \frac{1}{k} + \frac{1}{k(k-1)} \sum_{(\alpha,\beta) \in U_1} \Pr_{(\gamma,\delta) \xleftarrow{\$} U_2} [(\alpha, \beta) \sim (\gamma, \delta)] \\
&= \frac{1}{k} + \frac{R}{k(k-1)}
\end{aligned}$$

Splitting in the case when $\alpha = \beta$ and $\alpha \neq \beta$, where we know that the former happens in at most k cases, we can calculate

$$\begin{aligned}
R &= \sum_{(\alpha,\beta) \in U_1} \Pr_{(\gamma,\delta) \xleftarrow{\$} U_2} [(\alpha, \beta) \sim (\gamma, \delta)] \\
&\leq \sum_{\substack{(\alpha,\beta) \in U_1 \\ \alpha = \beta}} \Pr_{(\gamma,\delta) \xleftarrow{\$} U_2} [\gamma = \delta] + \sum_{\substack{(\alpha,\beta) \in U_1 \\ \alpha \neq \beta}} \Pr_{(\gamma,\delta) \xleftarrow{\$} U_2} [\alpha = \gamma \text{ and } \beta = \delta)] \\
&\leq k \cdot \frac{1}{k-1} + k(k-1) \cdot \frac{1}{k(k-1)} = \frac{2k-1}{k-1}.
\end{aligned}$$

It follows that

$$\Pr_{\{\pi_{i,b}\}} [h(x_{2,1} * h(x_{2,2}) = h(x_{1,1}) * h(x_{1,2})] \leq \frac{1}{k} + \frac{2k-1}{k(k-1)^2} = \frac{k}{(k-1)^2}.$$

□

Lemma 4. *Draw k-permutation hash functions on n input bits, $h_1, \ldots, h_t$ independently at random. Then the probability that there exists $\{x_{b,c}\}_{b,c \in \{1,2\}} \subseteq \{0,1\}^n$ with $x_{1,1} \neq x_{1,2}$, $x_{1,1} \neq x_{2,1}$, $x_{1,1} * x_{1,2} = x_{2,1} * x_{2,2}$, and $h_i(x_{1,1}) * h_i(x_{1,2}) = h_i(x_{2,1}) * h_i(x_{2,2})$ for every $i \in [t]$ is strictly less than $\frac{k^t \cdot 6^n}{(k-1)^{2t}}$.*

Proof. Let $u \in (\mathcal{Z}_2)^n$ be given and let s be the number of entries of u that are (a, a), denoting the equivalence classes of $\mathcal{Z}_2$ as before by $(0, 1)$, $(1, 0)$, and (a, a). The number of choices for $\{x_{b,c}\}_{b,c \in \{1,2\}} \subseteq \{0,1\}^n$ such that $u = x_{1,1} * x_{1,2} = *x_{2,1} * x_{2,2}$ is 2^{2s} since for all entries $i \in [n]$ such that $u_i = (0, 1)$ or $u_i = (1, 0)$, $x^i_{1,1} = x^i_{2,1}$ and $x^i_{1,2} = x^i_{2,2}$ are fixed and for all entries $i \in [n]$ such that $u_i = (a, a)$ we have $x^i_{1,1} = x^i_{2,1}$ and $x^i_{1,2} = x^i_{2,2}$, but they are not fixed. Now, note that the number of $u \in (\mathcal{Z}_2)^n$ with exactly s (a, a)-entries is $2^{n-s}\binom{n}{s}$. Summing over all possible u we get the total number of choices $\{x_{b,c}\}_{b,c \in \{1,2\}} \subseteq \{0,1\}^n$ with $x_{1,1} * x_{1,2} = x_{2,1} * x_{2,2}$ to be

$$\sum_{s=0}^{n} 2^{n-s} \binom{n}{s} 2^{2s} = 2^n \sum_{s=0}^{n} 2^s \binom{n}{s} = 6^n.$$

By Lemma 3 the probability that any single choice of $\{x_{b,c}\}_{b,c\in\{1,2\}} \subseteq \{0,1\}^n$ such that $x_{1,1} \neq x_{1,2}$, $x_{1,1} \neq x_{2,1}$ and $x_{1,1} * x_{1,2} = x_{2,1} * x_{2,2}$ satisfies $h_i(x_{1,1}) * h_i(x_{1,2}) = h_i(x_{2,1}) * h_i(x_{2,2})$ for all $i \in [t]$ is strictly less than $\left(\frac{k}{(k-1)^2}\right)^t$. Thus, our conclusion follows immediately by the union bound.

□

For the next theorem we define a function that combines several permutation hash functions into one. Choose k-permutation hash functions $h_1, \ldots, h_t$ as discussed above, and define the main hash function $h(x) = h_1(x) \mid\mid h_2(x) \mid\mid \cdots \mid\mid h_t(x)$.

Theorem 3. *If $t \geq \frac{(1+\log_2(3))n+\lambda}{2\log_2(k-1)-\log_2(k)}$ then the function h as defined above statistically secures every function $f\colon \{0,1\}^n \to \{0,1\}$.*

Proof. Fix k and n. By Lemma 4, h statistically secures every function if

$$\frac{k^t \cdot 6^n}{(k-1)^{2t}} \leq 2^{-\lambda}.$$

Taking logarithms on both sides, this is equivalent to

$$\log_2(k)t + (1+\log_2(3))n - 2t\log_2(k-1) \leq -\lambda.$$

Rearranging, this yields that h statistically secures every function for

$$t \geq \frac{(1+\log_2(3))n+\lambda}{2\log_2(k-1)-\log_2(k)}.$$

□

4.3 Implementing Permutation Hash Functions over CLT13

We now describe how to efficiently secure a branching program over CLT using permutation hash functions. We first describe how to construct a branching program that takes an input $u||v$ and checks whether $v = h_i(u)$ for a single hash function h_i from the previous section. We then describe a technique that allows evaluating branching programs in parallel as long as they have the same input function. Finally, we use this technique to efficiently add a securing permutation hash function to any matrix branching program over CLT13.

Implementing One Permutation Hash Function Check. Assume we are given a k-permutation hash function $h = h_i$ of input size n as in the previous section, with the corresponding permutations $\{\pi_{i,b}\}_{i,b}$. We construct a branching program BP^h over some $R \cong \mathbb{Z}_p$ that works over inputs in $\mathbb{Z}_{\vec{v}}$, where $\vec{v} = [2, \ldots, 2, k] \in \mathbb{Z}^{n+1}$. This branching program will compute h over the first n bits in the input and then check if the result matches the final piece of input.

In the following sections we denote a branching program of length l that works over inputs in $\mathbb{Z}_{\vec{v}}$ by the tuple $(\mathbf{mat}, M_1, M_2, \mathbf{inp})$, where $\mathbf{mat}(i)$ is an indexed family $\{M_{i,c}\}_{c\in\mathbb{Z}_{v_i}}$ for all $i \in [l]$. M_0 and M_{l+1} are "bookend" vectors. This branching program is evaluated over an input $x \in \mathbb{Z}_{\vec{v}}$ by computing the following product:

$$M_0 \times \prod_{i=1}^{n} M_{i,x_{\mathbf{inp}(i)}} \times M_{l+1}.$$

For a k-permutation hash function h, let

$$\mathsf{BP}^h = (\textbf{mat}^h, M_0^h, M_{n+2}^h, \textbf{inp}^h).$$

The components of BP^h are defined as follows:

1. $\textbf{mat}^h(1) = \{M_{1,c}^h\}_{c \in \mathbb{Z}_k}$, where $M_{1,c}^h \in M_k(R)$ is the permutation matrix corresponding to the transposition $(1\ c)$.
2. $\textbf{mat}^h(i) = \{M_{i,b}^h\}_{b \in \{0,1\}}$ for $2 \leq i \leq n+1$, where $M_{i,b}^h \in M_k(R)$ is the permutation matrix corresponding to $\pi_{i-1,b}$.
3. $M_0^h = [1, 0, \ldots, 0] \in R^k$.
4. $M_{n+2}^h = [0, 1, \ldots, 1]^T \in R^k$.
5. $\textbf{inp}^h(i) = \begin{cases} n+1 & i = 1 \\ i-1 & 2 \leq i \leq n+1 \end{cases}$

Consider an evaluation of the branching program BP^h over an input $u||v$, where $u \in \{0,1\}^n$ and $v \in \mathbb{Z}_k$. The result is of the form

$$M_0^h \times M_{1,v}^h \times \prod_{i=2}^{n+1} M_{i,u_{i-1}}^h \times M_{n+2}^h. \tag{3}$$

The product $\prod_{i=2}^{n+1} M_{i,u_{i-1}}^h \times M_{n+2}^h$ results in a column vector with a 0 at position $h(u)$ and 1s in every other position. The product of this result with $M_{1,v}^h$ produces $[0, 1, \ldots, 1]^T$ if and only if $h(u) = v$ and otherwise produces a vector with a 0 in a position other than the first and 1s everywhere else. Multiplying by M_0^h thus produces 0 if and only if $h(u) = v$. In conclusion, evaluating BP^h on input $u \mid\mid v$ outputs 0 if and only if $h(u) = v$.

Parallel Branching Programs. The CLT13 multilinear map uses a ring of composite order, which allows for a certain type of parallel branching program computation. Namely, we can construct a branching program where each step in actuality encodes steps for several branching programs, and the parent branching program evaluates to zero if and only if all of its underlying branching programs do. In this section we describe how to construct such a parallel computation.

Let n be the dimension of the CLT13 instantiation we are using, defined to be the number of prime factors of the ring order. (We assume it is squarefree.) Let $\mathsf{BP}_1, \mathsf{BP}_2, \ldots, \mathsf{BP}_n$ be the set of branching programs we want to evaluate in parallel. We make several assumptions restricting the types of branching programs that we can execute in parallel. First, assume they are all of the same length l and all take inputs from $\mathbb{Z}_{\vec{v}}$. Second, recall that the plaintext ring for a CLT13 instantiation is of the form $\mathbb{Z}_g \cong \bigoplus_{i=1}^n \mathbb{Z}_{g_i}$ for primes g_i. Assume the matrices of BP_i are defined over the field Z_{g_i} for all i. We also assume the matrices of all the BP_i are of the same size, which is without loss of generality since we can pad them with identity matrices. Finally, assume every BP_i has the same input function $\textbf{inp}$.

Let $\mathsf{BP}_i = (\textbf{mat}_i, M_{i,0}, M_{i,l+1}, \textbf{inp})$, where $\textbf{mat}_i(j) = \{M_{i,j,c}\}_{c \in \mathbb{Z}_{v_j}}$. We construct a new branching program $\mathsf{BP}' = (\textbf{mat}', M_0', M_{l+1}', \textbf{inp})$ over the ring $\mathbb{Z}_g$, where $\textbf{mat}'(j) = \{M_{j,c}'\}_{c \in \mathbb{Z}_{v_j}}$ with

$M'_{j,c} \equiv M_{i,j,c} \pmod{g_i}$ for all $i \in [n]$, $j \in [l]$, and $c \in \mathbb{Z}_{v_j}$, and additionally $M'_0 \equiv M_{i,0} \pmod{g_i}$ and $M'_{l+1} \equiv M_{i,l+1} \pmod{g_i}$ for all $i \in [n]$. If we evaluate the branching program BP' on $x \in \mathbb{Z}_{\vec{v}}$ as the product

$$M'_0 \times \prod_{j=1}^{l} M'_{j,x_{\mathsf{inp}(j)}} \times M'_{l+1} \pmod{g},$$

the result, $\mathsf{BP}'(x)$, is zero if and only if

$$M_{i,0} \times \prod_{j=1}^{l} M_{i,j,x_{\mathsf{inp}(j)}} \times M_{i,l+1} \equiv 0 \pmod{g_i}$$

for all $i \in [n]$.

Securing an Arbitrary Branching Program. Assume we have a branching program $\mathsf{BP} = (\mathbf{mat}, M_0, M_{l+1}, \mathbf{inp})$, $\mathbf{mat}(j) = \{M_{j,b}\}_{b\in\{0,1\}}$ over $\{0,1\}^n$ which we would like to secure. We need to construct a new branching program BP' which computes BP but also requires an additional section of input which should be a hash of the first part. BP' must check whether the hash is valid and must always return a nonzero value if it is not.

More formally, let u be an input to BP. Let $h_1, \ldots, h_t$ be k-permutation hash functions on $|u|$ bits. BP' takes input $u||v$ and checks whether $v_i = h_i(u)$ for all $i \in [t]$. If so BP' returns $\mathsf{BP}(u)$, and if not BP' returns some nonzero value.

Let h_i be implemented by the branching program

$$\mathsf{BP}^{h_i} = (\mathbf{mat}^{h_i}, M^{h_i}_{b_1}, M^{h_i}_{b_2}, \mathbf{inp}^{h_i}),$$

where $\mathbf{mat}^{h_i}(1) = \{M^{h_i}_{1,c}\}_{c\in\mathbb{Z}_k}$ and $\mathbf{mat}^{h_i}(j) = \{M^{h_i}_{j,b}\}_{b\in\{0,1\}}$ for $2 \leq i \leq n+1$. We need to modify this branching program so that instead of taking an input $u||v \in \mathbb{Z}_{[2,\ldots,2,k]}$ of length $n+1$, it takes an input $u||v \in \mathbb{Z}_{[2,\ldots,2,k,\ldots,k]}$ of length $n+t$ and checks whether $v_i = h_i(u)$. We can do this by altering the input function $\mathbf{inp}^{h_i}$ to set $\mathbf{inp}^{h_i}(1) = n+i$, but this would result in the branching programs BP^{h_i} having different input functions for different values of i, which is not compatible with parallel branching program evaluation. So instead we pad the branching program so that the first t entries are all the identity matrix except for the i'th entry which is $\{M^{h_i}_{1,c}\}_{c\in\mathbb{Z}_k}$. Then the input function can be set to be the same for all i. Specifically, we redefine $\mathbf{mat}^{h_i}$ as follows:

- $\mathbf{mat}^{h_i}(i) = \{M^{h_i}_{1,c}\}_{c\in\mathbb{Z}_k}$
- $\mathbf{mat}^{h_i}(j) = \{I_k\}_{c\in\mathbb{Z}_k}$ for all $1 \leq j \leq t, j \neq i$
- $\mathbf{mat}^{h_i}(j) = \{M^{h_i}_{j-t+1,b}\}_{b\in\{0,1\}}$ for all $t+1 \leq j \leq t+n$

and we redefine $\mathbf{inp}^{h_i}$ as follows:

$$\mathbf{inp}^{h_i}(j) = \mathbf{inp}^{h}(j) = \begin{cases} n+j & 1 \leq j \leq t \\ j-t & t+1 \leq j \leq t+n. \end{cases}$$

We are now ready to use parallel branching program evaluation to combine the hash function checks with the original branching program functionality. We use $t+1$ branching programs, one of which is a modified version of BP and the others are modified versions of the BP^{h_i}. Every modified branching program will have length $t+l$ and will share the same input function $\textbf{inp}'$, so as to facilitate parallel evaluation.

We first define the new input function:

$$\textbf{inp}'(j) = \begin{cases} n+j & 1 \leq j \leq t \\ \textbf{inp}(j-t) & t+1 \leq j \leq t+n \end{cases}$$

The reasoning for this definition will become clear shortly. We modify $\mathsf{BP} = (\textbf{mat}, M_0, M_{l+1}, \textbf{inp})$ by padding the branching program with identity matrices at the beginning while leaving the rest of the program unchanged. So $\textbf{mat}(j) = I$ for $1 \leq i \leq t$, and $\textbf{mat}(j) = \{M_{j-t,b}\}_{b\in\{0,1\}}$ for $t+1 \leq j \leq t+l$. Note that BP should now be evaluated using the input function $\textbf{inp}'$.

Finally we describe how we modify BP^{h_i} to work with the input function $\textbf{inp}'$. The problem with the definition of BP^{h_i} given above is that the input function during the latter part of the branching program, where $j > t$, does not match the input function of BP. We could fix this by padding BP with more identity matrices so that the computation of the h_i and the computation of BP would happen sequentially, but this would add to the total length of the resulting parallel branching program. Instead we make an observation about the computation of h_i which allows for some flexibility in how we define the input function to the program. We will use these observations to rearrange BP^{h_i} so that it matches $\textbf{inp}'$ exactly.

We observe that changing the order in which we read the input does not affect whether $h = h_1 \mid\mid \cdots \mid\mid h_t$ secures a function or not since this is equivalent to for each h_i to permute the order of composition of the permutations of h_i. Since each of the permutations of h_i are chosen uniformly at random, this does not affect the distribution of h_i.

Given this observation, we can redefine $\textbf{mat}_{h_i}$ as follows without changing its functionality. Let f_j be the smallest r such that $\textbf{inp}(r) = j$ (we assume that BP reads all of its input at some point such that this is well-defined). Then we set

- $\textbf{mat}^{h_i}(t+f_j) = \{M^{h_i}_{j+1,b}\}_{b\in\{0,1\}}$ for all $1 \leq j \leq n$.
- $\textbf{mat}^{h_i}(t+r) = \{I\}_{b\in\{0,1\}}$ for all $r \in [l] \setminus \{f_j\}_{j\in[n]}$.
- $\textbf{mat}^{h_i}(i) = \{M^{h_i}_{1,c}\}_{c\in\mathbb{Z}_k}$.
- $\textbf{mat}^{h_i}(j) = \{I_k\}_{c\in\mathbb{Z}_k}$ for all $1 \leq j \leq t, j \neq i$

Thus we have $t+1$ branching programs which now share the same input function, and evaluating the branching programs in parallel as described above achieves the functionality of BP' as desired.

References

[AGIS14] Prabhanjan Ananth, Divya Gupta, Yuval Ishai, and Amit Sahai. Optimizing obfuscation: Avoiding barrington's theorem. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 646–658, New York, NY, USA, 2014. ACM.

[AS17] Prabhanjan Ananth and Amit Sahai. Projective arithmetic functional encryption and indistinguishability obfuscation from degree-5 multilinear maps. In *Advances in Cryptology – EUROCRYPT 2017*, 2017.

[BGK+14] Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Protecting obfuscation against algebraic attacks. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 221–238. Springer, 2014.

[BISW17] Dan Boneh, Yuval Ishai, Amit Sahai, and David J. Wu. Lattice-based snargs and their application to more efficient obfuscation. In *Advances in Cryptology – EUROCRYPT 2017*, 2017.

[CGH+15] Jean-Sébastien Coron, Craig Gentry, Shai Halevi, Tancréde Lepoint, Hemanta K. Maji, Eric Miles, Mariana Raykova, Amit Sahai, and Mehdi Tibouchi. Zeroizing without low-level zeroes: New mmap attacks and their limitations. *LNCS*, 9215:247–266, 2015.

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

[CHL+15] Jung Hee Cheon, Kyoohyung Han, Changmin Lee, Hansol Ryu, and Damien Stehlé. Cryptanalysis of the multilinear map over the integers. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, pages 3–12, 2015.

[CLLT17] Jean-Sébastien Coron, Moon Sung Lee, Tancrède Lepoint, and Mehdi Tibouchi. Zeroizing attacks on indistinguishability obfuscation over CLT13. In *Public-Key Cryptography - PKC 2017 - 20th IACR International Conference on Practice and Theory in Public-Key Cryptography, Amsterdam, The Netherlands, March 28-31, 2017, Proceedings, Part I*, pages 41–58, 2017.

[CLT13] Jean-Sébastien Coron, Tancréde Lepoint, and Mehdi Tibouchi. Practical multilinear maps over the integers. *LNCS*, 8042:476–493, 2013.

[GGH13a] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 1–17. Springer, 2013.

[GGH+13b] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, pages 40–49. IEEE, 2013.

[GGH15] Craig Gentry, Sergey Gorbunov, and Shai Halevi. Graph-induced multilinear maps from lattices. In *Theory of Cryptography Conference*, pages 498–527. Springer, 2015.

[GLSW15] Craig Gentry, Allison Bishop Lewko, Amit Sahai, and Brent Waters. Indistinguishability obfuscation from the multilinear subgroup elimination assumption. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 151–170, 2015.

[GMM+16] Sanjam Garg, Eric Miles, Pratyay Mukherjee, Amit Sahai, Akshayaram Srinivasan, and Mark Zhandry. Secure obfuscation in a weak multilinear map model. In *Theory of Cryptography Conference*, pages 241–268. Springer, 2016.

[MSZ16] Eric Miles, Amit Sahai, and Mark Zhandry. Annihilation attacks for multilinear maps: Cryptanalysis of indistinguishability obfuscation over GGH13. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, pages 629–658, 2016.