2015 IEEE 56th Annual Symposium on Foundations of Computer Science

Indistinguishability Obfuscation from the Multilinear Subgroup Elimination Assumption

Craig Gentry
IBM Research
cbgentry@us.ibm.com

Allison Bishop Lewko[1]
Columbia University
allison@cs.columbia.edu

Amit Sahai[2]
UCLA
sahai@cs.ucla.edu

Brent Waters[3]
The University of Texas at Austin
bwaters@cs.utexas.edu

Abstract

We revisit the question of constructing secure general-purpose indistinguishability obfuscation, with a security reduction based on explicit computational assumptions over multilinear maps. Previous to our work, such reductions were only known to exist based on meta-assumptions and/or ad-hoc assumptions: In the original constructive work of Garg et al. (FOCS 2013), the underlying explicit computational assumption encapsulated an exponential family of assumptions for each pair of circuits to be obfuscated. In the more recent work of Pass et al. (Crypto 2014), the underlying assumption is a *meta-assumption* that also encapsulates an exponential family of assumptions, and this meta-assumption is invoked in a manner that captures the specific pair of circuits to be obfuscated. The assumptions underlying both these works substantially capture (either explicitly or implicitly) the actual structure of the obfuscation mechanism itself.

In our work, we provide the first construction of general-purpose indistinguishability obfuscation proven secure via a reduction to a natural computational assumption over multilinear maps, namely, the Multilinear Subgroup Elimination Assumption. This assumption does not depend on the circuits to be obfuscated (except for its size), and does not correspond to the underlying structure of our obfuscator. The technical heart of our paper is our reduction, which gives a new way to argue about the security of indistinguishability obfuscation.

I. Introduction

Program obfuscation has been a longstanding goal in cryptography, though for many years general solutions seemed elusive. Recently, this changed dramatically with the introduction of a general indistinguishability obfuscation ($i\mathcal{O}$) candidate by Garg, Gentry, Halevi, Raykova, Sahai, and Waters [GGH$^+$13b]. This candidate was constructed in two stages: First a candidate $i\mathcal{O}$ construction was given for $\mathbf{NC}^1$ circuits, and then it was proven that assuming the Learning With Errors (LWE) assumption, $i\mathcal{O}$ for $\mathbf{NC}^1$ circuits implies $i\mathcal{O}$ for all polynomial-size circuits. Subsequently, several papers have shown a wide range of cryptographic applications of $i\mathcal{O}$, including core primitives [SW14], functional encryption [GGH$^+$13b], deniable encryption [SW14], two round multi-party computation [GGHR14], non-interactive multi-party key exchange [BZ14], and many others.

The combination of an indistinguishability obfuscation candidate plus the demonstration of myriad cryptographic applications raises the possibility that $i\mathcal{O}$ could serve as a type of "central hub" - a foundation on which most cryptographic primitives, new and old, can be built. However, it is imperative that we gain greater confidence in the security of $i\mathcal{O}$ constructions for $\mathbf{NC}^1$ circuits. The works of [GGH$^+$13b], [BR14], [BGK$^+$13] made one kind of progress toward this goal by providing proofs of security in an idealized adversary model, where the adversary is limited to "generic multilinear" attacks.

However, ideally, we would like to follow the Golwasser-Micali [GM84] paradigm and base the security of $i\mathcal{O}$ on a believable assumption using a reduction in the standard model that considers arbitrary (computationally bounded) adversaries. Ideally, this assumption should be as different from our obfuscation techniques as possible, so the reduction carries significant information about how security is achieved.

Achieving this goal is the central focus of this paper. To this end, we provide a qualitatively new security reduction for a candidate $i\mathcal{O}$ construction that we believe significantly advances our understanding of the security of $i\mathcal{O}$. We reduce security to the Multilinear Subgroup Elimination Assumption, recently introduced by [GLW14]. Informally speaking, the Multilinear Subgroup Elimination Assumption can be seen as a natural analogue in the multilinear setting of the Subgroup Decision

[1]Supported by NSF CNS-1413971 and CCF-1423306.
[2]Research supported in part from a DARPA/ONR PROCEED award, a DARPA/ARL SAFEWARE award, NSF Frontier Award 1413955, NSF grants1228984, 1136174, 1118096, 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 U.S. Office of Naval Research under Contract N00014-11-1-0389. 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.
[3]Supported by NSF CNS-0952692, CNS-1228599 and CNS-1414082, DARPA SafeWare, Google Faculty Research award, the Alfred P. Sloan Fellowship, Microsoft Faculty Fellowship, and Packard Foundation Fellowship.

0272-5428/15 $31.00 © 2015 IEEE
DOI 10.1109/FOCS.2015.19

Assumption [BGN05] from the bilinear setting. An important feature of this assumption is that it does not directly deal with programs, or encompass the inner workings of obfuscation mechanisms, as we discuss further below.

Our assumption can be instantiated using an existing proposal for approximate multilinear maps [CLT15]. There are no known attacks on our assumption. We stress, however, that building instantiations of multilinear maps under various mathematical frameworks is a very active area of investigation, with multiple proposed candidates [GGH13a], [CLT13], [GGH15], [CLT15] where some of these candidates have been subject to notable cryptanalysis [GGH13a], [CHL$^+$15], [GHMS14], [BWZ14]. Our assumption, however, is quite natural, and we believe that future work on instantiating multilinear maps can be aided by the existence of our assumption as test-case for cryptanalysis[1].

Challenges.: To describe the challenges that must be overcome to prove the security of $i\mathcal{O}$ from different kinds of assumptions, we first give a brief overview of what $i\mathcal{O}$ security entails and how previous construction candidates have been analyzed. Informally, indistinguishability obfuscation requires that for any two program descriptions P_0 and P_1 from some class that are functionally equivalent (i.e. for all inputs x we have that $P_0(x) = P_1(x)$), no computationally bounded attacker can distinguish $i\mathcal{O}(P_0)$ from $i\mathcal{O}(P_1)$ with better than negligible advantage.

In all previous constructions of indistinguishability obfuscation based on concrete assumptions [GGH$^+$13b], [PST14], the security of the construction is based on families of assumptions (in the case of [PST14], such a family is called a meta-assumption) where the precise assumption being used depends quite intricately on the notion of indistinguishability obfuscation, embedding the actual programs to be obfuscated themselves into the assumption. Furthermore, these families of assumptions have also (explicitly or implicitly) embedded the actual structure of the $i\mathcal{O}$ construction into the assumption, thus "assuming away" a great deal of the computational hardness underlying the construction.

Our work: $i\mathcal{O}$ security from a natural assumption.: With this motivation, we set out to answer the following question:

Can we prove the security of indistinguishability obfuscation from a natural assumption over multilinear maps?

In fact, our result works with a natural sub-exponential hardness assumption over multilinear encodings recently introduced by [GLW14], that we call the Multilinear Subgroup Elimination Assumption, under which we can prove the security of our construction of indistinguishability obfuscation. This assumption makes no reference, explicitly or implicitly, to pairs of programs, or even programs at all. The assumption only depends on the maximum length of a matrix branching representation for the family of programs to be obfuscated. This assumption also provides a concrete target for the cryptanalysis of multilinear maps, since it focuses attention on a single assumption.

The Multilinear Subgroup Elimination Assumption states the following: In a composite-order k-multilinear setting, suppose there is one special prime factor c together with k distinguished prime factors $a_1, \ldots, a_k$, along with other prime factors. Then the assumption requires that given generators of all prime-order subgroups *except* for the subgroup of order c, and random elements of all composite-order subgroups that include c and exactly $k-1$ distinguished prime factors, the adversary cannot distinguish a random element in a composite-order subgroup of order $c \cdot a_1 \cdots a_k$, from a random element in a composite-order subgroup of order $a_1 \cdots a_k$. The formal statement of this assumption is given in Section III-C.

We prove security by means of a new reduction technique that yields significant insight into the nature of $i\mathcal{O}$. Critically, the new hybrid argument we develop allows us to isolate the "key" step at which the actual program switches in a new way: this isolation lets us deal with this key transformation in an information-theoretic manner, which departs fundamentally from previous reduction-based security arguments for $i\mathcal{O}$. Our new notion of input-activated obfuscation and our techniques for proving security of this notion constitute the technical heart of our paper. We elaborate on our techniques in Section I-A.

The 2^n security loss.: Our hybrid argument has a 2^n security loss, where n is the length of the inputs to the two programs P_0 and P_1. However, such a security loss seems inherent (this issue was first observed by [GGSW13] in the context of witness encryption) whenever we consider a natural (black-box) reduction to an instance-independent assumption. To see why such a security loss seems hard to avoid, let us suppose that we want to prove security of an $i\mathcal{O}$ scheme under an instance-independent assumption α, but without such a security loss. A (black-box) reduction of security must be able to take an attacker on a candidate $i\mathcal{O}$ system for any functionally equivalent pair P_0, P_1 of programs and turn it into an attacker on the underlying assumption α. However: any such reduction should confirm that the programs are functionally equivalent – something that in general should take roughly 2^n time (barring a major and unexpected advance in complexity theory). Otherwise, if the reduction does not confirm this, then it should also apply to programs that are almost functionally equivalent. As a result, one could use the reduction algorithm to directly and efficiently break the assumption: In particular, consider an attacker that first receives the challenge from the assumption and then chooses a program pair P_0, P_1 that have

[1]Indeed, the proposal for multilinear maps that we rely on [CLT15], arose in part to eliminate the cryptanalysis that broke the Multilinear Subgroup Elimination Assumption [GHMS14], [BWZ14] over an earlier proposed candidate [CLT13]

different outputs on some secret input x^*, where the secret x^* is known to the attacker, but x^* is hidden from the reduction inside the programs P_0, P_1. For instance, the attacker constructs a program P_b such that $P_b(x)$ outputs 0 unless x comes a sparse pseudorandom set S, such that the attacker knows an efficient method to sample $x \in S$. If x does lie in S, then $P_0(x)$ outputs 0, but $P_1(x)$ outputs 1. Note that P_0 and P_1 are almost equivalent, except on this sparse pseudorandom set S. The attacker then runs the reduction and gets a challenge obfuscated program P^*. The attack algorithm is trivially able to simulate a distinguisher between $P^* \leftarrow i\mathcal{O}(P_0)$ and $P^* \leftarrow i\mathcal{O}(P_1)$ by querying $P^*(x^*)$, where $x^* \in S$ is sampled by the adversary. It then continues to run the reduction algorithm, which eventually must break the assumption. This strategy will actually work in the case of a reduction algorithm that proceeds obliviously without learning whether the two program descriptions are indeed functionally equivalent. Therefore, it seems that such a reduction without a 2^n security loss is implausible[2].

Note that in contrast, a reduction to an exponentially large *family* of assumptions can essentially choose which assumption it wishes to use based on the program descriptions P_0, P_1. For example, in the case of [GGH$^+$13b], the assumption chosen simply embeds an actual obfuscation of one of the program descriptions. If one ever fed the reduction a pair of non-functionally equivalent programs, then the reduction would map this to a false assumption. Since this false assumption is by definition not in the family of assumptions, there is no contradiction. Similarly in [PST14], for a pair of non-functionally equivalent programs, the [PST14] reduction would attempt to invoke their meta-assumption on an underlying distribution for which (existentially) there is a generic attack – and the meta-assumption is defined so that it does not have to hold in this case. Thus, we can see how such a reduction can avoid the need for discovery of functional equivalence by simply "passing" the problem of dealing with problematic instances into the assumption description itself. As a result, strictly speaking, our assumption is incomparable to [GGH$^+$13b] or [PST14], since those works rely on a family of assumptions against polynomial-time adversaries, whereas our assumption requires security against sub-exponential adversaries.

Finally, we believe that because our new reduction technique more directly deals with the complexity introduced by obfuscation, it will likely have other implications. Indeed, thinking more speculatively, we believe that our reduction ideas are the most likely to set us on an eventual path to a reduction under a classic assumption such as (sub-exponentially secure) LWE. At the same time, getting such a reduction appears quite challenging at the moment, as it remains unknown how to emulate the key features of multilinear maps that we need using only LWE.

Subsequent Work.: Since the initial announcement of our work, our techniques and ideas have influenced several papers. For example, our construction ideas and proof techniques played a crucial role in the work of [GGHZ14], who showed how to build fully secure functional encryption from assumptions over multilinear maps. More recently, two independent groups [AJ15], [BV15] showed how to construct indistinguishability obfuscation from subexponentially secure compact functional encryption schemes, and were in part inspired by our ideas (as explicitly noted for the use of subexponential security by [AJ15]). As of now, these recent results do not actually provide an alternative method for building indistinguishability obfuscation on simpler assumptions, since compact functional encryption can at present only be based on indistinguishability obfuscation [GGH$^+$13b]. However, we are excited to see that our work has played an important role in the further development of the security of indistinguishability obfuscation.

A. Our Techniques: An intuitive Overview

Let us first step back, and think about our central goals when building a reduction to argue the security of $i\mathcal{O}$. Recall that $i\mathcal{O}$ demands that $i\mathcal{O}(P_0)$ is indistinguishable from $i\mathcal{O}(P_1)$ whenever P_0 and P_1 are functionally equivalent. In any hybrid argument that proceeds from $i\mathcal{O}(P_0)$ to $i\mathcal{O}(P_1)$, apparently there will be some critical hybrid step(s) in which some *actual* underlying computation switches in some way from "something like P_0" to "something like P_1." In all previous works based on explicit computational assumptions [GGH$^+$13b], [PST14], the assumption itself was invoked to handle this case: In [GGH$^+$13b], the $i\mathcal{O}$ property itself followed directly from the assumption. Clearly, if we wish to avoid this, we need to find some other way of dealing with this critical moment when some computation shifts from P_0 to P_1. For inspiration, we look to the original argument on the security of obfuscation in a generic security model [GGH$^+$13b]: this generic proof does not contain a reduction at all, so it may seem (and it indeed mostly is) of little use to us. However, this generic proof identifies a powerful information-theoretic[3] manner of dealing with the shift from P_0 to P_1 as it manifests itself in the generic security argument, specifically in the context of the obfuscated computation on *a single input*. This is done by means of Kilian's simulation [Kil88]. We begin by asking, can we invoke this information-theoretic argument in the context of a reduction?

In order to do so, we will need to establish a hybrid argument that essentially changes the computation from P_0 to P_1 input-by-input, and thereby isolates the critical computation to a single input. As such, there will be a number of technical

[2]However, formalizing this intuition remains an important open question.

[3]We note that the fact that this argument is information-theoretic is not necessarily crucial. It is, however, very convenient.

hurdles – most notably, we need the obfuscation scheme itself to be decomposable into 2^n variations that somehow isolate each individual input.

A trivial "straw man" idea for allowing our obfuscation to be decomposable into 2^n variations would be to have 2^n separate parallel copies of the obfuscator, where only one is "active" for each possible input. In fact, this idea of parallel copies of obfuscation, each of which is active only on certain inputs, does turn out to be quite useful intuitively. However, clearly we cannot afford a 2^n blowup in the actual obfuscation size. So instead, we must find a succinct way to decompose the space of 2^n possible inputs into a polynomial set of different buckets, which cover all 2^n inputs[4]. Each such bucket will be associated with the partial obfuscation of one particular program P, where $P = P_0$ or $P = P_1$, but there will be many different buckets of inputs that will be associated with the same program. The obfuscation can be partial, because each such obfuscation only has to work with the set of active inputs associated with the bucket to which it is connected. Crucially, we will want to move to a collection of buckets where there is one particular bucket that only corresponds to a single active input x^*. We will maintain the invariant that *only* when there is at most a single active input x^* isolated in a bucket, do we switch the associated program from $P = P_0$ to $P = P_1$. As we discussed above, the key argument of this step will turn out to be information-theoretic in nature, thus letting us address the thorny issue of how to move from an obfuscation of one program to the obfuscation of another[5].

Another primary motivation of this invariant is what it guarantees for every other hybrid step in our argument: In other hybrid steps, we will need to transfer inputs from one bucket to another, so that we can proceed to isolate some other input $\hat{x}^*$. But when we transfer some inputs from one bucket to another, critically we only move inputs from buckets associated with a program P to other buckets that are associated with the same program P. This means that the hybrid arguments associated with such transfers of inputs between buckets intuitively *do not care* about hiding any program. They can hold with respect to adversaries that fully know which programs are associated with which buckets. As such, we can design algebraic (computational) reduction techniques for all of these hybrids, based on assumptions that do not need to be tailored to specific programs.

The above discussion is intended to give some intuitive "flavor" of our techniques. We now proceed to provide a more technical guide to our techniques.

B. Our Techniques: A technical guide

To implement the ideas we sketched above, we introduce and implement a critical abstraction that we call *input-activated obfuscation*.

Input-Activated Obfuscation: the technical heart of our work.: At the heart of our work is the implementation of the kind of "parallelized" obfuscation that we described above, that we formally call input-activated obfuscation. We stress that our construction and security analysis of input-activated obfuscation depart fundamentally from all previous works about reduction-based proofs for obfuscation [PST14] or weakenings of obfuscation such as witness encryption [GLW14]. For example, for witness encryption, the work of [GLW14] only had to deal with DNF formulas, and for DNF formulas there is always (at least) a single clause that witnesses the fact that a particular input fails to satisfy the DNF formula. This observation was at the core of the approach of [GLW14], influencing both their cryptographic design and proof techniques. DNF formulas are (as far as we know) only a tiny subset, within $\mathbf{AC}^0$, of all $\mathbf{NC}^1$ circuits that we need to obfuscate. Therefore, our construction and approach to proving security of input-activated obfuscation takes a different approach, since the approach of [GLW14] simply would not suffice for us.

Informally speaking, in input-activated obfuscation, there are several different programs being obfuscated in a single obfuscated object, and each of these programs are only active for certain inputs. As sketched above, in this abstraction, roughly speaking, there will be two kinds of security properties:

- **Activating/Deactivating inputs.** We will need to argue that it is not possible for an adversary to distinguish an obfuscation where certain "redundant" inputs are activated or deactivated for a particular program P, as long as they are already active for another copy of P. (This security property corresponds to the case of "transferring inputs" between programs discussed above.) We emphasize that, by design, in this security property, the actual set of obfuscated programs does not change. Therefore, to establish this security property by means of a security reduction, the reduction can have full knowledge of actual programs being obfuscated.

[4]For building intuition, it is useful to think of these buckets as forming a partition of all 2^n inputs. However, in our actual argument, there will be hybrids where an input is in multiple buckets (in our actual proof, these "buckets" correspond to "columns of an input-activated matrix"). However, the set of inputs in buckets associated with P_0 will always be disjoint from the set of inputs in buckets associated with P_1.

[5]We do also employ other non-information-theoretic arguments even when dealing with a bucket with only a single active input, but these arguments are intuitively about how to securely "zero out" parts of an obfuscation that are only needed for inactive inputs.

- **Single-Input Program Switching.** We will also need to argue that if one of our programs P is only active for a single input x, then it is indistinguishable to an adversary to switch this program P with any other program $P*$ such that $P(x) = P*(x)$.

Constructing this notion of input-activated obfuscation is the central challenge of our paper. As mentioned above, we will want to build it so that the Single-Input Program Switching property can be proven by invoking Kilian's information-theoretic argument to replace P with $P*$.

Defining and Constructing Input-Activated Obfuscation.: The core of our abstraction of input-activated obfuscation will be a $n \times \ell \times 2$ matrix M with entries in $\{0,1\}$, and an ordered set of ℓ programs, $P = (P_1, P_2, \ldots, P_\ell)$ from a specified family of programs $\{\mathcal{P}_\lambda\}$, parameterized by a security parameter λ. We abuse standard terminology slightly by saying that such a matrix M has n rows and ℓ columns. We say that every row-column pair (i, j) where $i \in [n], j \in [\ell]$ has two associated "slots," denoted by $M_{i,j,0}$ and $M_{i,j,1}$.

We will call the matrix M an *input-activated matrix*. This is because for each column j of an input-activated matrix M, we define a corresponding boolean function $f_j : \{0,1\}^n \to \{0,1\}$, where program P_j is active on input x iff $f_j(x) = 1$. These functions f_j are defined as follows:

$$f_j(x) = \begin{cases} 1, & \text{if } M_{i,j,x_i} = 1 \text{ for all } i \in [n]; \\ 0, & \text{otherwise.} \end{cases}$$

We will sometimes abuse terminology and say that column j of an input-activated matrix itself evaluates to 1 on an input x iff $f_j(x) = 1$.

An input-activated obfuscation scheme consists of two algorithms:

- **Creation.** The creation algorithm $Create(\lambda, M, P)$ takes in a security parameter λ, an $n \times \ell \times 2$ input-activated matrix M, and an ordered set $P = (P_1, P_2, \ldots, P_\ell)$ of ℓ programs from $\mathcal{P}_\lambda$. It produces an input-activated obfuscation T.
- **Evaluation.** The evaluation algorithm $Eval(T, x \in \{0,1\}^n)$ takes in an input-activated obfuscation T and an n-bit input x, and outputs 0 or 1.

Note that because input-activated obfuscation deals with the parallel obfuscation of several programs simultaneously, even the correctness property requires some care to define.

Correctness requires the following: Consider any input $x \in \{0,1\}^n$. We let $S_x \subseteq [\ell]$ denote the set of column indices j that are active on input x, i.e. such that $f_j(x) = 1$. Then if $S_x \neq \emptyset$ and $P_j = P_{j'}$ for all $j, j' \in S_x$, we require that $Eval(T, x) = P_j(x)$ for all $j \in S_x$. (This must hold for all inputs x.)

Observe that correctness only imposes a restriction on the output of the evaluation algorithm in certain cases. Indeed, we take care that these are the only cases that will arise in our use of the input-activated obfuscation abstraction.

The two informal security properties defined above are actually captured by four separate security games between a challenger and an attacked (see Section II for details).

Our construction of input-activated obfuscation from multilinear maps based on the Multilinear Subgroup Elimination Assumption is found in Section III. As outlined in our intuition above, Kilian's information theoretic argument will play a starring role in implementing the Single-Input Program Switching security property. This will be made possible because the parallel obfuscations of the programs $P_1, \ldots, P_\ell$ are performed in distinct subgroups. Thus, they are separately randomized, and Kilian's information-theoretic argument will still apply. Implementing the security properties that cover Activating/Deactivating Inputs is trickier. In these security games, a single entry in the input-activation matrix M is to be toggled (corresponding to Activating/Deactivating the inputs that rely on this entry being 1). The main idea behind achieving this type of security is to use the Multilinear Subgroup Elimination Assumption to allow us to move between hybrids where certain parts of the program are either activated, if the assumption gave us a challenge element whose order included the special prime factor c, to hybrids where these parts of the program are inactivated, if the assumption gave us a challenge element whose order did not include the special prime factor c. An essential challenge to achieving this is to be able to generate the remaining components of the input-activated obfuscation using only components given by the Multilinear Subgroup Elimination Assumption. This is accomplished with great care, including a redundant method of encoding the input-activation matrix with "helper" group elements we call *enforcing* elements.

Using Input-Activated Obfuscation to build $i\mathcal{O}$.: To use input-activated obfuscation to build $i\mathcal{O}$, we follow the high-level strategy outlined earlier here. In order to do so, we must use the local Activation/Deactivation security properties of input-activated obfuscation to transfer inputs from one copy of a program to another. We accomplish this by defining another abstraction that we call Positional Indistinguishability Obfuscation (see Section IV), and following a traversal that allows us to transfer inputs, using the Activation/Deactivation security properties, in such a way that eventually isolates every possible input. This part of our paper is technically straightforward (albeit notationally cumbersome), and there are many traversals

that can work. We choose a traversal that is a natural generalization of one already explicitly considered by [GLW14] for dealing with Positional Witness Encryption, and for completeness we recreate the adapted proof in Section V.

II. Input-Activated Obfuscation

In this section, we will describe an abstraction that we call *Input-Activated Obfuscation* ($ia\mathcal{O}$). This will function as the central abstraction of this work that will be used to connect to indistinguishability obfuscation via positional $i\mathcal{O}$ on one end and be realized by multilinear encodings on the other end. We will first present this abstraction in this section. We follow with our realization of Input-Activated Obfuscation via multilinear encodings in Section III. Next, we present positional $i\mathcal{O}$ in Section IV and show how it implies $i\mathcal{O}$. Then we show that Input-Activated Obfuscation implies positional $i\mathcal{O}$ in Section V.

The core of this abstraction will be a $n \times \ell \times 2$ matrix M with entries in $\{0,1\}$, and an ordered set of ℓ programs, $P = (P_1, P_2, \ldots, P_\ell)$ from a specified family of programs $\{\mathcal{P}_\lambda\}$, parameterized by a security parameter λ. We abuse standard terminology slightly by saying that such a matrix M has n rows and ℓ columns. We say that every row-column pair (i, j) where $i \in [n], j \in [\ell]$ has two associated "slots," denoted by $M_{i,j,0}$ and $M_{i,j,1}$.

We will call the matrix M an *input-activated matrix*. This is because for each column j of an input-activated matrix M, we define a corresponding boolean function $f_j : \{0,1\}^n \to \{0,1\}$, where program P_j is active on input x iff $f_j(x) = 1$. These functions f_j are defined as follows:

$$f_j(x) = \begin{cases} 1, & \text{if } M_{i,j,x_i} = 1 \text{ for all } i \in [n]; \\ 0, & \text{otherwise.} \end{cases}$$

We will sometimes abuse terminology and say that column j of an input-activated matrix itself evaluates to 1 on an input x iff $f_j(x) = 1$.

An input-activated obfuscation scheme consists of two algorithms:

Creation.: The creation algorithm $Create(\lambda, M, P)$ takes in a security parameter λ, an $n \times \ell \times 2$ input-activated matrix M, and an ordered set $P = (P_1, P_2, \ldots, P_\ell)$ of ℓ programs from $\mathcal{P}_\lambda$. It produces an input-activated obfuscation T.

Evaluation.: The evaluation algorithm $Eval(T, x \in \{0,1\}^n)$ takes in an input-activated obfuscation T and an n-bit input x, and outputs 0 or 1.

These two algorithms should satisfy several properties. Note that because $ia\mathcal{O}$ deals with the parallel obfuscation of several programs simultaneously, even the correctness property requires some care to define.

Correctness.: We define perfect correctness as follows. Consider an input $x \in \{0,1\}^n$. We let $S_x \subseteq [\ell]$ denote the set of column indices j that are active on input x, i.e. such that $f_j(x) = 1$. Then if $S_x \neq \emptyset$ and $P_j = P_{j'}$ for all $j, j' \in S_x$, we require that $Eval(T, x) = P_j(x)$ for all $j \in S_x$. (This must hold for all inputs x.)

Observe that correctness only imposes a restriction on the output of the evaluation algorithm in certain cases. Indeed, we take care that these are the only cases that will arise in our use of the $ia\mathcal{O}$ abstraction.

We next define security properties for an input-activated obfuscation scheme. We define each property in terms of a game between a challenger and an attacker.

Inter-column Security Game.: This game is parameterized by a security parameter λ, an $n \times \ell \times 2$ input-activated matrix M, an ordered set of programs $P = (P_1, \ldots, P_\ell)$ in $\mathcal{P}_\lambda$, two column indices j and k in $[\ell]$, a row index i^* in $[n]$, and a slot index $\beta \in \{0,1\}$ such that $M_{i^*,j,\beta} = 1$. We require that $P_j = P_k$. By this, we mean that *the program descriptions must be identical*. Note that this is a more stringent requirement than simply saying they must agree on all inputs. We further require the following conditions on the j^{th} and k^{th} columns of M. For every row i and slot $\gamma \in \{0,1\}$, *except for* $i = i^*$ *and* $\gamma = 1 - \beta$, if $M_{i,k,\gamma} = 1$, then $M_{i,j,\gamma} = 1$ as well. When this condition holds, we say that column j *dominates* column k, although strictly speaking this is not required in slot $1 - \beta$ of row i^*. All of these parameters are given both to the challenger and to the attacker.

The challenger samples a uniformly random bit $b \in \{0,1\}$. If $b = 0$, it runs $Create(\lambda, M, P)$ to produce an input-activated obfuscation T. If $b = 1$, it forms M' by copying M except for flipping just one entry: $M'_{i^*,k,\beta} = 1$ if $M_{i^*,k,\beta} = 0$, and $M'_{i^*,k,\beta} = 0$ if $M_{i^*,k,\beta} = 1$. It then runs $Create(\lambda, M', P)$ to produce T. The challenger gives T to the attacker, who finally must guess the value of the bit b.

Note that because we require $M_{i^*,j,\beta} = 1$, the change imposed by the inter-column game does not change the actual set of inputs that are active across the union of columns j and k. This is because any inputs that become active or become inactive on column k when $M_{i^*,k,\beta}$ changes are active in column j.

Definition II.1. *We say an input-activated obfuscation scheme has* inter-column security *if for every polynomial attacker $\mathcal{A}$, there exists a negligible function $negl(\lambda)$ such that the attacker's advantage in the Inter-Column Game is $\leq negl(\lambda)$, for any valid settings of M, P, j, k, i^*, β.*

Intra-column Security Game.: This game is parameterized by a security parameter λ, a $n \times \ell \times 2$ input-activated matrix M, an ordered set of programs $P = (P_1, \ldots, P_\ell)$, an index j of a column in M such that there is some row i^* where both slots take the value 0, and an alternate column $C \in \{0,1\}^{n \times 2}$ such that the i^* row also has both slots equal to 0. All of these parameters are given both to the challenger and to the attacker.

The challenger samples a uniformly random bit $b \in \{0,1\}$. If $b = 0$, it runs $Create(\lambda, M, P)$ to produce T. If $b = 1$, it forms M' by replacing the j^{th} column of M with C, and then runs $Create(\lambda, M', P)$ to produce T. It gives T to the attacker, who must then guess the value of the bit b.

Definition II.2. *We say an input-activated obfuscation scheme has* intra-column security *if for every polynomial attacker $\mathcal{A}$, there exists a negligible function $negl(\lambda)$ such that the attacker's advantage in the Intra-column Game is $\leq negl(\lambda)$, for any valid settings of M, P, j, C.*

Completely Inactive Program Security Game.: This game is parameterized by a security parameter λ, a $n \times \ell \times 2$ input-activated matrix M, an ordered set of programs $P = (P_1, \ldots, P_\ell)$, an alternate program P^*, and an index $j \in [\ell]$ such that the j^{th} column of M contains all zero entries. All of these parameters are given both to the challenger and to the attacker.

The challenger samples a uniformly random bit $b \in \{0,1\}$. If $b = 0$, it runs $Create(\lambda, M, P)$ to produce T. If $b = 1$, it forms P' by modifying P to replace the j^{th} program with P^*, and then runs $Create(\lambda, M', P')$ to produce T. It gives T to the attacker, who must then guess the value of the bit b.

Definition II.3. *We say an input-activated obfuscation scheme has* completely inactive program security *if for every polynomial attacker $\mathcal{A}$, there exists a negligible function $negl(\lambda)$ such that the attacker's advantage in the Completely Inactive Program Security Game is $\leq negl(\lambda)$, for any valid settings of M, P, P^*, j.*

Single-input Program Switching Security Game.: This game is parameterized by a security parameter λ, a $n \times \ell \times 2$ input-activated matrix M, an ordered set of programs $P = (P_1, \ldots, P_\ell)$, an alternate program P^*, and an index $j \in [\ell]$ such that the j^{th} column of M corresponds to a point function f_j evaluating to 1 on a single input x^* (and evaluating to 0 on all other inputs), with $P^*(x^*) = P_j(x^*)$. All of these parameters are given both to the challenger and to the attacker.

The challenger samples a uniformly random bit $b \in \{0,1\}$. If $b = 0$, it runs $Create(\lambda, M, P)$ to produce T. If $b = 1$, it forms P' by modifying P to replace the j^{th} program with P^*, and then runs $Create(\lambda, M', P')$ to produce T. It gives T to the attacker, who must then guess the value of the bit b.

Definition II.4. *We say an input-activated obfuscation scheme has* single-input program switching security *if for every polynomial attacker $\mathcal{A}$, there exists a negligible function $negl(\lambda)$ such that the attacker's advantage in the Single-input Program Switching Security Game is $\leq negl(\lambda)$, for any valid settings of M, P, P^*, j.*

III. An Instantiation in a Model of Composite Order Multilinear Groups

We now present an instantiation of input-activated obfuscation in a model of composite order multilinear groups. Our construction will produce "program-carrying" group elements that reflect the matrices of the associated branching programs in their exponents, as well as "enforcing" group elements which will play a role in enforcing that the input bits used in the evaluation of the branching program are consistent, and also for enforcing the input activation properties implied by the input-activated matrix. For oblivious matrix branching programs of width 5 and length z, there will be $5 \cdot 5 \cdot 2 \cdot z$ program-carrying group elements, naturally corresponding to the $2z$ matrices of dimension 5×5 that form such a branching program. For inputs of the length n, there will be $2n$ enforcing elements (each indexed by an $i \in [n]$ and a bit b). Our group will be $(z + n)$-linear.

An honest evaluation of the construction will first take the program-carrying elements corresponding to the matrices that match the desired input and multiply them in the exponent of the multilinear group. It will then extract a single entry that will be zero or nonzero depending on the program output. It then further applies the multilinear map with the n enforcing elements corresponding to the input (i.e. one for each i, with the bit value matching the i^{th} bit of the input).

We will use many subgroups of distinct prime orders to serve separate purposes in our construction. There will be a prime q_j for each column index $j \in [\ell]$, and the evaluation of the program P_j will happen in the exponent of this subspace. There will also be primes $p_1, \ldots, p_{z+n}$ that will add randomness to the partial computations of the construction, preventing an attacker from learning anything that it should not learn by putting together certain combinations of elements that would never occur in an honest evaluation. More precisely, we can partition the group elements into $z + n$ classes - z classes of program-carrying elements, one for each step of the branching program, and n classes of enforcing elements (one for each input bit). An honest evaluation always performs $z+n$ multilinear computations that respect this class structure. These

additional primes $p_1, \ldots, p_{z+n}$ will prevent a meaningful result with deviation from this structure occurs, for example when two elements of the same class are used together in the multilinear map.

Finally, there are primes $r_{s,b}$ for each $s \in [z]$ and $b \in \{0,1\}$. Essentially, the prime $r_{s,b}$ will be used to enforce that if an evaluation uses the bit value b at step s of the branching program, then it must use the same bit value b for the corresponding input-enforcing element, otherwise there will be a random contribution from the subgroup of order $r_{s,b}$ that is never canceled out, hence obscuring the final evaluation result.

Before proceeding to the details of our construction, we give a quick review of the formal model for composite order multilinear groups.

A. An Abstract Model of Composite Order Multilinear Groups

In this section, we will work with an abstract model of symmetric, composite order multilinear groups. We let G denote a cyclic group of composite order $N = p_1 p_2 \cdots p_r$ (where $p_1, \ldots, p_r$ are distinct primes). We suppose that G comes equipped with an efficiently computable, non-degenerate k-linear map $E : G^k \to G_T$, that is actually implemented in a graded manner. We let $1_G, 1_{G_T}$ denote the respective identity elements in G and G_T. For each prime p_i, we let g_{p_i} denote a generator of the subgroup of order p_i inside G. We note that whenever $h \in G$ belongs to the subgroup of order $p_1 \cdots p_{i-1} p_{i+1} \cdots p_r$ inside G and $g_2, \ldots, g_{k-1} \in G$ are arbitrary, we have that

$$E(h, g_2, \ldots, g_{k-1}, g_{p_i}) = 1_{G_T}.$$

More generally, the various prime order subgroups inside G are "orthogonal" under the multilinear map, meaning that a particular prime order subgroup only contributes to the result of E if there are non-trivial components in this subgroup on *every* input to E. This is a simple consequence of the definition of multilinearity.

We let $\mathcal{G}(\lambda, r, k)$ denote a group generation algorithm that takes in a security parameter λ, a number of prime factors r, and a level of multilinearity k and outputs a description of such a group G. We assume the description includes the group order N, the individual primes $p_1, \ldots, p_r$, a generator g of G, and efficient algorithms for the group operations in G and G_T as well as E. We note that with g and the individual primes, one can produce a generator for any particular subgroup of G.

B. Construction

We now construct an input-activated obfuscation scheme in this setting. Our construction will proceed in two phases. First we construct a preliminary scheme that we prove satisfies inter-column security, single-input program switching security, and completely inactive program security. Next we apply a simple transformation (analogous to the one described in [GLW14]) to obtain a variant that also satisfies intra-column security. Our preliminary scheme is:

Create$_{lite}(\lambda, M, P)$*::* This algorithm takes in a security parameter λ, an $n \times \ell \times 2$ input-activated matrix M, and an ordered set $P = (P_1, \ldots, P_\ell)$ of programs from P_λ. We require that all P_j are oblivious matrix branching programs of length z for inputs of length n, represented as z pairs of 5×5 matrices $A_{j,t,b}$ (where $t \in [z]$ and $b \in \{0,1\}$). The only differences between the P_j's are expressed in the matrix contents, not the length or the mapping of input bits to steps of the branching program (this is guaranteed by the obliviousness property). For each t in $[z]$, we let $\alpha(t) \in [n]$ denote the index of the input bit being referenced at that step.

It first generates a $(z+n)$-linear group G of composite order

$$N = q_1 \cdots q_\ell \cdot p_1 \cdots p_{z+n} \prod_{s=1}^{z} r_{s,0} \cdot r_{s,1}.$$

These are all distinct primes, and we let g_{q_j}, for example, denote a random generator of the subgroup of order q_j. We choose uniformly random matrices $R_1, \ldots, R_{z-1}$ in $Z_N^{5\times 5}$. We note that these are distributed uniformly and independently modulo each prime factor of N by the Chinese Remainder Theorem. We also note that these are invertible modulo each prime with all but negligible probability. For each $j \in [\ell]$, $t \in [z]$, and $b \in \{0,1\}$, we define $B_{j,t,b} = R_{t-1}^{-1} A_{j,t,b} R_t$, where R_0, R_z are defined to be the identity matrix. We note, for use later, that for any setting of $j, k \in [\ell]$, with any $t \in [z]$ and $b \in \{0,1\}$, that if $A_{j,t,b} = A_{k,t,b}$, then $B_{j,t,b} = B_{k,t,b}$. This is true since the R_i matrices are defined globally over Z_N, even though this choice induces a set of independent random matrices over each prime factor of N.

We will produce an input-activated obfuscation T containing $5 \cdot 5 \cdot 2z + 2n$ elements of G, each corresponding to an entry of a matrix in the branching program or to an index (y, b) where $y \in [n], b \in \{0,1\}$. We will refer to the group elements corresponding to matrix entries as "program-carrying elements" and those corresponding to indices (y, b) as "enforcing elements."

To generate a program-carrying group element $g_{w,v,t,b}$ for the (w,v) entry of the matrix at a step $t \in [z]$ in the branching program, corresponding to bit value $b \in \{0,1\}$, and input index $i = \alpha(t) \in [n]$, we proceed as follows. For each j from 1 to ℓ, we compute a group element $u_j = g_{q_j}^{B_{j,t,b}(w,v)}$. We also sample random elements $\gamma_1, \ldots, \gamma_{t-1}, \gamma_{t+1}, \ldots, \gamma_{z+n}$, where for all $i \in [z+n] \setminus \{t\}$, we sample γ_i from the subgroup of order p_i.

Finally, for each $s \in [z], b' \in \{0,1\}$, we sample a group element $d_{s,b'}$ randomly in the subgroup of order $r_{s,b'}$, except when $s = t$ and $b' = 1 - b$. In this case, we set $d_{t,1-b} = 1$.

We then define

$$g_{w,v,t,b} = \gamma_1 \cdots \gamma_{t-1}\gamma_{t+1} \cdots \gamma_{z+n} \prod_{j=1}^{\ell} u_j \prod_{s,b'} d_{s,b'}.$$

To generate an enforcing group element $g_{y,b}$ for $y \in [n]$ and $b \in \{0,1\}$, we first sample random elements $\gamma_1, \ldots, \gamma_{z+y-1}, \gamma_{z+y+1}, \ldots, \gamma_{z+n}$, where for all $i \in [z+n] \setminus \{z+y\}$, we sample γ_i from the subgroup of order p_i.

For each j from 1 to ℓ, we sample a group element u_j as follows. If $M_{y,j,b} = 1$, we sample u_j uniformly at random from the subgroup of order q_j. If $M_{y,j,b} = 0$, we set $u_j = 1$. For each s, b', we sample $d_{s,b'}$ as follows. If $\alpha(s) \neq y$ or $b' \neq b$, then $d_{s,b'}$ is sampled uniformly at random from the subgroup of order $r_{s,b'}$. Otherwise, when $\alpha(s) = y, b' = b$, we set $d_{s,b} := 1$.

We then define

$$g_{y,b} = \gamma_1 \cdots \gamma_{z+y-1}\gamma_{z+y+1} \cdots \gamma_{z+n} \prod_{j=1}^{\ell} u_j \prod_{s,b'} d_{s,b'}.$$

We output the collections of elements $g_{w,v,t,b}$ and $g_{y,b}$ as the input-activated obfuscation T.

$Eval_{lite}(T, x)$:: For an input $x \in \{0,1\}^n$ and an input-activated obfuscation $T = \{g_{w,v,t,b}, g_{y,b}\}$, we evaluate as follows. For each t, b, we let $k_{t,b}$ denote the 5×5 collection of elements $\{g_{w,v,t,b}\}$, as w, v range over $[5]$. We view $k_{t,b}$ as a 5×5 matrix of group elements. We extend the multilinear map to matrices of group elements in the natural way, by computing the matrix product in the exponent. We can then compute:

$$e(k_{1,x_{\alpha(1)}}, k_{2,x_{\alpha(2)}}, \ldots, k_{z,x_{\alpha(z)}}), \tag{1}$$

which will be a 5×5 collection of group elements in G_z.

We will then choose a single non-diagonal entry. We choose this entry so that it is non-zero as an entry of the product $A_{j,1,x_{\alpha(1)}} A_{j,2,x_{\alpha(2)}} \cdots A_{j,z,x_{\alpha(z)}}$ whenever the branching program outputs 0 (i.e. whenever this matrix product is not the identity matrix). We call this single group element k. We then compute:

$$e(k, g_{1,x_1}, \ldots, g_{n,x_n}) \in G_T. \tag{2}$$

We apply the zero test to the result of (2). If the zero test detects a zero, we output 1. Otherwise, we output 0.

Correctness.: We first observe that the subgroups of order $p_1, \ldots, p_z$ do not contribute to the result of (1), since each subgroup of order p_t is absent from the matrix elements in $k_{t,x_{\alpha(t)}}$. Similarly, the subgroups of order $p_{z+1}, \ldots, p_{z+n}$ do not contribute to the result of (2). For each (s,b), if $b \neq x_{\alpha(s)}$, then the subgroup of order $r_{s,b}$ does not contribute because it is absent from the matrix elements in $k_{s,x_{\alpha(s)}}$. If $b = x_{\alpha(s)}$, then the subgroup of order $r_{s,b}$ does not contribute because it is absent from the element $g_{\alpha(s),x_{\alpha(s)}}$.

Thus the only potential contributions to the value of (1) come from the subgroups of order $q_1, \ldots, q_\ell$. For each q_j, we observe that we will get precisely the matrix product $A_{j,1,x_{\alpha(1)}} A_{j,2,x_{\alpha(2)}} \cdots A_{j,z,x_{\alpha(z)}}$ modulo q_j in the exponent of (1). If the input-activated matrix has a 0 in the slot corresponding to x_i on some row i in column j, then one of the included enforcing elements will cause the q_j subgroup to be absent, and hence we will ultimately not get a contribution. If instead the j^{th} column is activated by the input x, this contribution in (1) will be multiplied by non-zero terms in (2), and this will be 0 in the tested entry if and only if the branching program P_j outputs 0 on the input x.

We will prove in the following subsection that the scheme Create_{lite}, Eval_{lite} satisfies inter-column, completely inactive program switching, and single-input program switching security. To obtain a scheme that also satisfies intra-column security, we define algorithms:

Create(λ, M, P):: This algorithm takes in a security parameter λ, an $n \times \ell \times 2$ input-activated matrix M, and an ordered set $P = (P_1, \ldots, P_\ell)$ of programs from P_λ. It forms an input-activated matrix M' that is $n \times (\ell + n\ell) \times 2$ by appending $n\ell$ new columns to M. For each $j \in [\ell]$, there will be n new columns appended (all with associated program P_j), with the i^{th} one having 0's in both slots of row i and 1's everywhere else. We let P' denote the resulting (expanded) ordered program list. Then $\text{Create}_{lite}(\lambda, M', P')$ is called to produce the output T.

Eval(T, x)*::* For an input $x \in \{0,1\}^n$ and an input-activated obfuscation $T = \{g_{w,v,t,b}, g_{y,b}\}$, we simply run $\text{Eval}_{lite}(T, x)$.

We note that correctness for Create and Eval follows immediately from correctness for Create_{lite} and Eval_{lite}, as the additional columns are not activated on any inputs.

C. The Multilinear Subgroup Elimination Assumption

We now describe our computational assumption, which was previously used in [GLW14].

The (μ, ν)-multilinear subgroup elimination assumption: This assumption is parameterized by positive integers μ, and ν. It concerns a μ-linear group of order $N = a_1 \dots a_\mu b_1 \dots b_\nu c$, where $a_1, \dots, a_\mu, b_1, \dots, b_\nu, c$ are $\mu + \nu + 1$ distinct primes. We give out generators $g_{a_1}, \dots, g_{a_\mu}, g_{b_1}, \dots, g_{b_\nu}$ for each prime order subgroup *except for the subgroup of order* c. For each $i \in [\mu]$, we also give out a group element h_i sampled uniformly at random from the subgroup of order $ca_1 \cdots a_{i-1} a_{i+1} \cdots a_\mu$. The challenge term is a group element $T \in G$ that is either sampled uniformly at random from the subgroup or order $ca_1 \cdots a_\mu$ or uniformly at random from the subgroup of order $a_1 \cdots a_\mu$. The task is to distinguish between these two distributions of T.

D. Security Properties

Lemma III.1. *For $\mu := z + n$ and $\nu := \ell + 2z - 1$, the (μ, ν)-multilinear subgroup elimination assumption implies inter-column security for our algorithms Create_{lite}, Eval_{lite} in Section III-B.*

Proof: We suppose there exists a PPT attacker $\mathcal{A}$ who achieves a non-negligible advantage in the inter-column game for some valid setting of M, P, j, k, i, β. We will create a PPT attacker $\mathcal{B}$ that achieves a non-negligible advantage in breaking the (μ, ν)-multilinear subgroup elimination assumption. $\mathcal{B}$ is given $g_{a_1}, \dots, g_{a_\mu}, g_{b_1}, \dots, g_{b_\nu}$, $h_1, \dots, h_\mu$, and a challenge term T. Its task is to guess whether T was sampled from the subgroup of order $ca_1 \cdots a_\mu$ or the subgroup of order $a_1 \cdots a_\mu$.

$\mathcal{B}$ implicitly sets $q_k = c$, and $p_t = a_t$ for every $t \neq z + i$. It sets $q_j = a_{z+i}$. It bijectively maps the remaining p_{z+i}, $q_{j'}$'s and $r_{s,b}$'s to the primes $b_1, \dots, b_\nu$. Then $\mathcal{B}$ samples the randomized matrices $B_{j,t,b}$ for all $j \in [\ell]$, $t \in [z]$, and $b \in \{0,1\}$.

To make a group element $g_{w,v,t,b}$, $\mathcal{B}$ proceeds as follows. It can use the subgroup generators at its disposal to sample γ_1, …, γ_{t-1}, γ_{t+1}, …, γ_{z+n}, $u_{j'}$ for all $j' \neq j, k$, and all of the $d_{s,b'}$ elements appropriately. It computes:

$$g_{w,v,t,b} := h_t^{B_{j,t,b}(w,v)} \gamma_1 \cdots \gamma_{t-1} \gamma_{t+1} \cdots \gamma_{z+n} \prod_{j' \neq j,k} u_{j'} \prod_{s,b'} d_{s,b'},$$

where the matrix $B_{j,t,b} = B_{k,t,b}$ (modulo N) because $A_{j,t,b} = A_{k,t,b}$. Note that h_t is nontrivial in both the subgroup of order $c = q_k$ and the subgroup of order $a_{z+i} = q_j$, and thus this term is distributed as it should be.

To make a group element $g_{y,b}$, $\mathcal{B}$ proceeds as follows. It can directly sample γ_1, …, γ_{z+y-1}, γ_{z+y+1}, …, γ_{z+n} and $d_{s,b'}$ elements appropriately by using the given subgroup generators. It can similarly sample elements $u_{j'}$ for $j' \neq j, k$. When $y \neq i$, one of two cases must occur. In the first case, $M_{y,k,b} = 0$, and thus we can set $u_k = 1$. We sample u_j appropriately, using the given generator $g_{a_{z+i}}$ if $M_{y,j,b} = 1$. Then, $\mathcal{B}$ can compute:

$$g_{y,b} = \gamma_1 \cdots \gamma_{z+y-1} \gamma_{z+y+1} \cdots \gamma_{z+n} \prod_{j'=1}^{\ell} u_{j'} \prod_{s,b'} d_{s,b'}.$$

In the second case, $M_{y,k,b} = 1$, which implies that $M_{y,j,b} = 1$ by the conditions underlying the inter-column game. In this case, $\mathcal{B}$ selects a random exponent $\alpha \in \mathbb{Z}_N$ and computes:

$$g_{y,b} = h_{z+y}^{\alpha} \gamma_1 \cdots \gamma_{z+y-1} \gamma_{z+y+1} \cdots \gamma_{z+n} \prod_{j' \neq j,k} u_{j'} \prod_{s,b'} d_{s,b'}.$$

Recall that h_{z+y} is nontrivial in both the subgroup of order $c = q_k$ and the subgroup of order $a_{z+i} = q_j$, since $y \neq i$. Thus, since the random choice of $\alpha \in \mathbb{Z}_N$ induces independent random values (α mod c) and (α mod a_{z+i}), these terms are distributed correctly.

When $y = i$ and $b = 1 - \beta$, there are two cases to consider. If $M_{i,k,b} = 0$, then we first sample u_j appropriately, using the given generator $g_{a_{z+i}}$ if $M_{y,j,b} = 1$. Then, $\mathcal{B}$ computes:

$$g_{i,b} = \gamma_1 \cdots \gamma_{z+y-1} \gamma_{z+y+1} \cdots \gamma_{z+n} \prod_{j' \neq k} u_{j'} \prod_{s,b'} d_{s,b'}.$$

On the other hand, if $M_{i,k,b} = 1$, then $\mathcal{B}$ selects a random exponent $\alpha \in \mathbb{Z}_N$ and samples u_j appropriately according to $M_{i,j,b}$. It then computes:

$$g_{i,b} = h_{z+i}^{\alpha} \gamma_1 \cdots \gamma_{z+y-1} \gamma_{z+y+1} \cdots \gamma_{z+n} \prod_{j' \neq k} u_{j'} \prod_{s,b'} d_{s,b'}.$$

Recall that h_{z+i} is trivial in the subgroup of order $a_{z+i} = q_j$, and thus the h_{z+i}^{α} term does not disturb the correctly sampled u_j term.

Finally, when $y = i$ and $b = \beta$, we must have $M_{i,j,\beta} = 1$, and $\mathcal{B}$ computes:

$$g_{i,\beta} = T \gamma_1 \cdots \gamma_{z+y-1} \gamma_{z+y+1} \cdots \gamma_{z+n} \prod_{j' \neq j,k} u_{j'} \prod_{s,b'} d_{s,b'}.$$

If T was sampled from the subgroup of order $ca_1 \cdots a_\mu$, this will be distributed as if $M_{i,k,\beta} = 1$. Otherwise, it will be distributed as if $M_{i,k,\beta} = 0$. Thus, $\mathcal{B}$ can leverage $\mathcal{A}$'s non-negligible advantage in the inter-column game to achieve a non-negligible advantage against the multilinear subgroup elimination assumption.

■

In our proof of single-input program switching security, we will use an information-theoretic lemma due to Kilian [Kil88]:

Lemma III.2. *[Kil88] Let x denote a single input to a matrix branching program $\{A_{t,b}\}$. Then if matrices $R_1, \ldots, R_{z-1}$ are chosen to be random invertible matrices, the distribution of the matrices $\{R_{t-1}^{-1} A_{t,x_{\alpha(t)}} R_t\}$ depends only on the output of the branching program evaluated on x.*

Essentially, this means that when what the attacker sees only involves one matrix from each position in the branching program, only the final output is information-theoretically revealed.

Lemma III.3. *For $\mu := z + n$ and $\nu := \ell + 2z - 1$, the (μ, ν)-multilinear subgroup elimination assumption implies single-input program switching security for our algorithms Create$_{lite}$, Eval$_{lite}$ in Section III-B.*

Proof: We will prove this via a hybrid argument that incrementally "erases" the branching program matrices not corresponding to the single relevant input (we will call it x^*) in the relevant subgroup using the subgroup elimination assumption. Once we have done this, we can argue information-theoretically to switch the programs and then reverse the hybrid to insert the new matrices. To execute this strategy, we begin by defining the following hybrid experiments. Exp$_0$ will denote the original game with the challenge bit set to 0 (here the original program P_j is used for the j^{th} column). For f from 1 to z, we define Exp$_f$ to be like Exp$_0$, except for the first f positions of the branching program, the corresponding program-carrying group elements for the bit values disagreeing with x^* will have no components in the q_j subgroup. (Note that in Exp$_z$, only one matrix will appear in the q_j subgroup per slot.)

We first argue that for each such f, Exp$_f$ is indistinguishable from Exp$_{f-1}$, under the multilinear subgroup elimination assumption. We suppose there is some $f \in [z]$ such that some PPT attacker $\mathcal{A}$ has a non-negligible advantage in distinguishing Exp$_f$ from Exp$_{f-1}$. We will use $\mathcal{A}$ to build a PPT attacker $\mathcal{B}$ to break the multilinear subgroup elimination assumption.

$\mathcal{B}$ is given $g_{a_1}, \ldots, g_{a_\mu}, g_{b_1}, \ldots, g_{b_\nu}$, $h_1, \ldots, h_\mu$, and a challenge term T. Its task is to guess whether T was sampled from the subgroup of order $ca_1 \cdots a_\mu$ or the subgroup of order $a_1 \cdots a_\mu$. It implicitly sets $q_j = c$, and $p_t = a_t$ for all $t \neq f$. It sets $r_{f,b^*} = a_f$, where b^* is the bit indicating the matrix to be erased (so b^* is the complement of the activated input bit for the f^{th} step of the branching program). The remaining primes, namely p_f, the $q_{j'}$ for $j' \neq j$, and the remaining $r_{s,b'}$ are mapped bijectively to the primes $b_1, \ldots, b_\nu$. Then, $\mathcal{B}$ samples the randomized matrices $B_{j,t,b}$ for all $j \in [\ell]$, $t \in [z]$, and $b \in \{0, 1\}$.

For any fixed $w, v \in [5]$, $t \in [z]$, and $b \in \{0, 1\}$, to make the group element $g_{w,v,t,b}$, the reduction $\mathcal{B}$ proceeds as follows. It can use the subgroup generators at its disposal to sample $\gamma_1, \ldots, \gamma_{t-1}, \gamma_{t+1}, \ldots, \gamma_{z+n}$, together with the $u_{j'}$ for all $j' \neq j$, and all of the $d_{s,b'}$ elements appropriately.

If $b = x^*_{\alpha(t)}$, or if $b = 1 - x^*_{\alpha(t)}$ but $t > f$, then $\mathcal{B}$ sets:

$$g_{w,v,t,b} = h_t^{B_{j,t,b}(w,v)} \gamma_1 \cdots \gamma_{t-1} \gamma_{t+1} \cdots \gamma_{z+n} \prod_{j' \neq j} u_{j'} \prod_{s,b'} d_{s,b'}.$$

Recall that when setting $g_{w,v,t,b}$, it should be that only $d_{t,1-b} = 1$, while all other $d_{s,b'}$ should be random in the subgroup of order $r_{s,b'}$. Therefore, we observe that our setting of $g_{w,v,t,b}$ is properly distributed when $t \neq f$ because the component in subgroup r_{f,b^*} should be random here, since $f \neq t$ (so it's acceptable that this component appears in h_t). To see it is also properly distributed when $t = f$, we note that when $t = f$ then to be in this case it must be that $b = 1 - b^*$, and therefore the component in subgroup r_{f,b^*} should be, and is, 1 here (as this component is missing from h_f).

The other case when $t = f$ is when $b = b^*$. To make g_{w,v,f,b^*}, $\mathcal{B}$ computes:

$$g_{w,v,f,b^*} = T^{B_{j,f,b^*}(w,v)} \gamma_1 \cdots \gamma_{f-1}\gamma_{f+1} \cdots \gamma_{z+n} \prod_{j' \neq j} u_{j'} \prod_{s,b'} d_{s,b'}.$$

We note the random component in the subgroup of order r_{f,b^*} appearing in the challenge term T is acceptable, as the d_{f,b^*} is supposed to be random for this group element.

Finally, if $b = 1 - x^*_{\alpha(t)}$ and $t < f$, then $\mathcal{B}$ simply sets

$$g_{w,v,t,b} = \gamma_1 \cdots \gamma_{t-1}\gamma_{t+1} \cdots \gamma_{z+n} \prod_{j' \neq j} u_{j'} \prod_{s,b'} d_{s,b'}.$$

Recall that for $t < f$, previous hybrids have already eliminated the q_j terms, and thus this is the correct distribution.

To make a group element $g_{y,b}$, the reduction $\mathcal{B}$ proceeds as follows. It can directly sample $\gamma_1, \ldots, \gamma_{z+y-1}, \gamma_{z+y+1}, \ldots, \gamma_{z+n}$ and $d_{s,b'}$ elements appropriately by using the given subgroup generators. It can similarly sample elements $u_{j'}$ for $j' \neq j$. If $M_{y,j,b} = 0$, then since u_j should equal 1, it can then set

$$g_{y,b} = \gamma_1 \cdots \gamma_{z+y-1}\gamma_{z+y+1} \cdots \gamma_{z+n} \prod_{j' \neq j} u_{j'} \prod_{s,b'} d_{s,b'}.$$

If $M_{y,j,b} = 1$, it must be the case that either $\alpha(f) \neq y$ or $b^* \neq b$. This is because column j, row y of M can only have one entry equal to 1 (since exactly one input is activated), and the entry $M_{\alpha(f),j,1-b^*} = 1$ by definition of b^*. This means that for these y, b, the reduction $\mathcal{B}$ can choose a random exponent $\alpha \in \mathbb{Z}_N$ and set

$$g_{y,b} = h^{\alpha}_{z+y}\gamma_1 \cdots \gamma_{z+y-1}\gamma_{z+y+1} \cdots \gamma_{z+n} \prod_{j' \neq j} u_{j'} \prod_{s,b'} d_{s,b'}.$$

The presence of an random r_{f,b^*} component on h_{z+y} is not problematic here, as $\alpha(f) \neq y$ or $b^* \neq b$ holds. Recall that in the enforcing elements $g_{y,b}$, the $r_{s,b'}$ components are to be random if $\alpha(s) \neq y$ or $b' \neq b$.

Thus, if T has a random $c = q_j$ component, then $\mathcal{B}$ has properly simulated Exp_{f-1}. If not, it has properly simulated Exp_f. This allows us to argue that under this multilinear subgroup elimination assumption, Exp_0 is computationally indistinguishable from Exp_z.

Next, we argue that the distribution of Exp_z is statistically close to the distribution of an Exp^*_z with P_j replaced by P^*, where P^* is any other branching program of the same length and input access pattern that agrees with P_j on the single activated input. We stress that this transition from Exp_z to Exp^*_z is information-theoretic (does not rely on any computational assumption).

First, we note that the uniform distribution of the randomizing matrices $R_1, \ldots, R_{z-1}$ (reduced modulo q_j) in Exp_z is only a negligible statistical distance from the distribution required for Lemma III.2, where they are sampled to be invertible (this is because a random matrix modulo q_j is invertible with all but negligible probability). So up to this negligible statistical distance, we have that the distribution of the randomized branching program matrices $B_{j,t,b}$ depends only on $P_j(x^*) = P^*(x^*)$. This means that the distribution of Exp^*_z is negligibly close to the distribution of Exp_z.

Once we are at Exp^*_z, we can apply the same hybrid steps in reverse to restore the matrices in the q_j subgroup of the program-carrying group elements, this time with P^* instead of P_j. This completes our proof of single-input program switching security.

■

Lemma III.4. *For $\mu := z+n$ and $\nu := \ell+2z-1$, the (μ, ν)-multilinear subgroup elimination assumption implies completely inactive program security for algorithms Create$_{lite}$, Eval$_{lite}$ in Section III-B.*

Proof: This follows from the same hybrid argument used in the proof of Lemma III.3, except that we "erase" the q_j subgroup components on *all* of the program-carrying group elements. (We note that the case in the above proof where $M_{y,j,b} = 1$ no longer arises.) Once we have erased all such components, we can reverse the process and iteratively insert the new program P^*. Here we do not need the information-theoretic argument from Killian, as we are able to erase all the matrices, leaving no distribution that needs to be matched. ■

Theorem III.5. *For $\mu := z+n$ and $\nu := \ell+n\ell+2z-1$, the (μ, ν)-multilinear subgroup elimination assumption implies intra-column security, inter-column security, completely inactive program switching security, and single-input program switching security for our algorithms Create, Eval in Section III-B.*

Proof: We note that inter-column security, completely inactive program switching security, and single-input program switching security for Create, Eval follow immediately from Lemmas III.1, III.3, and III.4, as the columns of the matrix M' that is input to Create_{lite} are a superset of the columns of the original M input to Create.

For intra-column security of Create, Eval, we will derive this as a consequence of the inter-column security of Create_{lite}, Eval_{lite}. We consider an instance of the intra-column game where column j of M has a row i^* with both slots set to 0 and we seek to replace this column C (which also has 0's in row i^*). For this, we use the appended column of M' that has the same program P_j and has 0's in row i^* and 1's everywhere else. Note that the changes we need to make to reach C are in slots where this appended column has 1's, and moreover its 1's cover all the 1's of the current j^{th} column as well as C. Thus, by iteratively applying inter-column security, we can change the j^{th} column of M to C. ■

IV. Positional Indistinguishability Obfuscation

We will first give our definition of positional indistinguishability obfuscation system. Then we show how it implies (standard) indistingusihability obfuscation by a hybrid argument.

We define a *positional indistinguishability obfuscation* scheme for for a program description class $\{\mathcal{P}_\lambda\}$ with inputs of size $n(\lambda)$. (For ease of exposition we will sometimes refer to $n(\lambda)$ as just n when it is clear from context.) For our purposes a program description [6] will be an encoding of a computable function. The system consists of two algorithms:

Obfuscation. The algorithm $\textit{Obfuscate}_{\text{PIO}}(1^\lambda, P_0, P_1, t)$ takes as input a security parameter 1^λ, two program descriptions $P_0, P_1 \in \{\mathcal{P}_\lambda\}$, and a position index $t \in [0, 2^n]$ and outputs an obfuscated program description P.

Evaluation. The algorithm $\textit{Eval}_{\text{PIO}}(P, x)$ takes as input a program description P (constructed from program description class $\{\mathcal{P}_\lambda\}$) and a length n input x and gives an output in the image of $\{\mathcal{P}_\lambda\}$.

Given an input string $x \in \{0,1\}^n$ we will sometimes slightly abuse notation and also refer to x as an integer in $[0, 2^n - 1]$ where the most significant bit is the leftmost bit. In other words, we consider the integer $x = \Sigma_{i=1}^{n} x_i \cdot 2^{n-1}$, where x_i is the i-th bit of the string x. Similarly, sometimes we will abuse notation in the other way, and consider an integer $x \in [0, 2^n - 1]$ to be a string.

Definition IV.1 ((Perfect) Correctness of Positional Indistinguishability Obfuscation)**.** *For any security parameter λ, any pair of program descriptions $P_0, P_1 \in \{\mathcal{P}_\lambda\}$ and for any input $x \in \{0,1\}^n$ we have that*

$$\textit{Eval}_{\text{PIO}}\big(\textit{Obfuscate}_{\text{PIO}}(1^\lambda, P_0, P_1, t), x\big) = \begin{cases} P_0(x) & \text{if } x \geq t \\ P_1(x) & \text{if } x < t. \end{cases}$$

A. Security of Positional Indistinguishability Obfuscation

The security of positional indistinguishability obfuscation is given in terms of three security properties.

Program Description Hiding A.: The first property is parameterized by two program descriptions P_0, P_1. Informally, the security property states that if one encrypts to the "first' position $t = 0$ (where n is the input length for program description class $\{\mathcal{P}_\lambda\}$) that no attacker can distinguish whether an obfuscated program description P is a positional obfuscation to the pair of program descriptions (P_0, P_1) or (P_0, P_0). Intuitively, this is because there is no input that will actually "use" P_1.

We define the (parameterized) advantage of an attacker as

$$\mathsf{A} : \mathsf{HidingPIO}\ \mathsf{Adv}_{\mathcal{A}, P_0, P_1}(\lambda) =$$

$$\Pr[\mathcal{A}(\textit{Obfuscate}_{\text{PIO}}(1^\lambda, P_0, P_1, t = 0)) = 1] - \Pr[\mathcal{A}(\textit{Obfuscate}_{\text{PIO}}(1^\lambda, P_0, P_0, t = 0)) = 1].$$

Definition IV.2 (Program Description Hiding A Security of Positional Indistinguishability Obfuscation)**.** *We say that a positional indistinguishability obfuscation for program description class $\{\mathcal{P}_\lambda\}$ is Program Hiding A secure if for any probabilistic poly-time attack algorithm $\mathcal{A}$ there exists a negligible function in the security parameter $\textit{negl}(\cdot)$ such that for all program description pairs $P_0, P_1 \in \{\mathcal{P}_\lambda\}$ we have* $\mathsf{A} : \mathsf{HidingPIO}\ \mathsf{Adv}_{\mathcal{A}, P_0, P_1}(\lambda) \leq \textit{negl}(\lambda)$.

Program Description Hiding B.: The second property is very similar to the first except it is stated for the opposite end ($t = 2^n$) of the index spectrum. Here no attacker can distinguish whether an obfuscated program description P is a positional obfuscation to the pair of program descriptions (P_0, P_1) or (P_1, P_1).

We define the (parameterized) advantage of an attacker as

$$\mathsf{B} : \mathsf{HidingPIO}\ \mathsf{Adv}_{\mathcal{A}, P_0, P_1}(\lambda) =$$

$$\Pr[\mathcal{A}(\textit{Obfuscate}_{\text{PIO}}(1^\lambda, P_0, P_1, t = 2^n)) = 1] - \Pr[\mathcal{A}(\textit{Obfuscate}_{\text{PIO}}(1^\lambda, P_1, P_1, t = 2^n)) = 1].$$

[6] We will sometimes use the terms program and program description interchangeably.

Definition IV.3 (Program Description Hiding B Security of Positional Indistinguishability Obfuscation)**.** *We say that a positional indistinguishability obfuscation for program description class $\{\mathcal{P}_\lambda\}$ is Program Description Hiding B secure if for any probabilistic poly-time attack algorithm $\mathcal{A}$ there exists a negligible function in the security parameter $negl(\cdot)$ such that for all program description pairs $P_0, P_1 \in \{\mathcal{P}_\lambda\}$ we have* $\mathsf{B} : \mathsf{HidingPIO\ Adv}_{\mathcal{A},P_0,P_1}(\lambda) \leq negl(\lambda)$.

Position Indistinguishability.: The third, and most significant, security game is positional indistinguishability. Informally, this security game states that it is hard to distinguish between a positional obfuscation to position t from an encryption to $t+1$ when the program descriptions P_0, P_1) have the same output on input t: i.e. $P_0(t) = P_1(t)$. Positional indistinguishability security is parameterized by program descriptions $P_0, P_1 \in \{\mathcal{P}_\lambda\}$ and a position $t \in [0, 2^n - 1]$ where n is the input length. We define the (parameterized) advantage of an attacker as

$$\mathsf{PosPIO\ Adv}_{\mathcal{A},P_0,P_1,t}(\lambda) =$$

$$\Pr[\mathcal{A}(\textit{Encrypt}_{\mathrm{PWE}}(1^\lambda, P_0, P_1, t+1)) = 1] - \Pr[\mathcal{A}(\textit{Encrypt}_{\mathrm{PWE}}(1^\lambda, P_0, P_1, t)) = 1].$$

Definition IV.4 (Position Indistinguishability Security of Positional Indistinguishability Obfuscation)**.** *We say that a positional indistinguishability obfuscation scheme for program description class $\{\mathcal{P}_\lambda\}$ is Position Indistinguishability secure if for any probabilistic poly-time attack algorithm $\mathcal{A}$ there exists a negligible function in the security parameter $negl(\cdot)$ such that for all $P_0, P_1 \in \{\mathcal{P}_\lambda\}$, and any $t \in [0, 2^n - 1]$ where $P_0(t) = P_1(t)$ we have* $\mathsf{PosPIO\ Adv}_{\mathcal{A},P_0,P_1,t}(\lambda) \leq negl(\lambda)$.

We let $\mathsf{PosPIO\ Adv}_{\mathcal{A},P_0,P_1}(\lambda)$ be the maximum value of $\mathsf{PosPIO\ Adv}_{\mathcal{A},P_0,P_1,t}(\lambda)$ over $t \in [0, 2^n]$ for each λ.

B. Building Indistinguishability Obfuscation from Positional Indistinguishability Obfuscation

We now describe how to build indistinguishability obfuscation from positional indistinguishability obfuscation, in a rather simple way. To obfuscate a program description $P \in \{\mathcal{P}_\lambda\}$ simply do a positional indistinguishability obfuscation to position $t = 0$ and use P as the inputs for both program descriptions.

Like for positional witness encryption, the proof comes from a hybrid security argument. Consider two program descriptions $P_0, P_1 \in \{\mathcal{P}_\lambda\}$ that are different, but are functionally equivalent, I.e. for all $x \in [0, 2^n - 1]$: $\quad P_0(x) = P_1(x)$. Then no attacker can distinguish an obfuscation to position t to one of $t + 1$. This argument is repeatedly applied to "move" the encryption position from 0 to 2^n. The cost of performing this hybrid is a security factor of 2^n and thus it innately requires complexity leveraging. The utilization of the abstraction is that each individual step of the hybrid is only concerned with whether the two program descriptions agree on *one* particular input. This isolation will eventually lead to security from instance independent assumptions.

We now formally describe the construction of indistinguishability obfuscation from positional indistinguishability obfuscation. We follow with a security proof.

$i\mathcal{O}(1^\lambda, P)$ calls $\textit{Obfuscate}_{\mathrm{PIO}}(1^\lambda, P, P, t = 0)$.

$\textit{Eval}(P, x)$ calls $\textit{Eval}_{\mathrm{PIO}}(P, x)$.

The correctness of the indistinguishability obfuscation system follows immediately from the correctness properties of positional indistinguishability obfuscation.

We now state and prove our the security theorem.

Theorem IV.5. *Consider the constructed indistinguishability obfuscation scheme for a program description class $\{\mathcal{P}_\lambda\}$ and the indistinguishability property for any pairs of program descriptions $P_0, P_1 \in \{\mathcal{P}_\lambda\}$ where $P_0(x) = P_1(x)\ \forall x$. We have that for any polynomial time attacker $\mathcal{A}$*

$$\mathsf{IO\ Adv}_{\mathcal{A},P_0,P_1}(\lambda) \leq 2^n \cdot \mathsf{PosPIO\ Adv}_{\mathcal{A},P_0,P_1}(\lambda) + 2^n \cdot \mathsf{PosPIO\ Adv}_{\mathcal{A},P_1,P_1}(\lambda) +$$
$$\mathsf{A} : \mathsf{HidingPIO\ Adv}_{\mathcal{A},P_0,P_1}(\lambda) + \mathsf{B} : \mathsf{HidingPIO\ Adv}_{\mathcal{A},P_0,P_1}(\lambda).$$

Proof:

We now give prove the theorem by a simple hybrid argument. The hybrid sequence defines a how an obfuscated program description is generated. We enumerate the hybrid steps below to for program descriptions P_0, P_1 where $P_0(x) = P_1(x)\ \forall x$.

- $\mathsf{Hyb}_{\mathrm{start}}$: The first hybrid generates the obfuscated program description as:

$$\textit{Obfuscate}_{\mathrm{PIO}}(1^\lambda, P_0, P_0, t = 0)$$

We observe that this is defined to be an obfuscation of P_0 as $i\mathcal{O}(1^\lambda, P_0)$.

- $\mathsf{Hyb}_{\mathsf{i}}$ for $i \in [0, 2^n]$: In $\mathsf{Hyb}_{\mathsf{i}}$ the obfuscated program description is generated as

$$\mathit{Obfuscate}_{\mathrm{PIO}}(1^\lambda, P_0, P_1, t = i)$$

  We observe that for any algorithm $\mathcal{A}$, its advantage in distinguishing between $\mathsf{Hyb}_{\mathrm{start}}$ and Hyb_0 can be at most $\mathsf{A} : \mathsf{HidingPIO}\ \mathsf{Adv}_{\mathcal{A},P_0,P_1}(\lambda)$. In addition, for all $i \in [0, 2^n - 1]$, the advantage of $\mathcal{A}$ in distinguishing between $\mathsf{Hyb}_{\mathsf{i}}$ and $\mathsf{Hyb}_{\mathsf{i}+1}$ can be at most $\mathsf{PosPIO}\ \mathsf{Adv}_{\mathcal{A},P_0,P_1}(\lambda)$. This follows from the fact that $P_0(i) = P_1(i)$. It follows that the advantage in distinguishing between Hyb_0 and Hyb_{2^n} is at most $2^n \cdot \mathsf{PosPIO}\ \mathsf{Adv}_{\mathcal{A},P_0,P_1}(\lambda)$.
- $\mathsf{Hyb}'_{\mathsf{i}}$ for $i \in [0, 2^n]$: In $\mathsf{Hyb}'_{\mathsf{i}}$ the obfuscated program description is generated as

$$\mathit{Obfuscate}_{\mathrm{PIO}}(1^\lambda, P_1, P_1, t = i)$$

  Note that the proof will proceed through these hybrids in the reverse order, starting with Hyb'_{2^n} and proceeding to Hyb'_0. We observe that for any algorithm $\mathcal{A}$, its advantage in distinguishing between Hyb_{2^n} and Hyb'_{2^n} can be at most $\mathsf{B} : \mathsf{HidingPIO}\ \mathsf{Adv}_{\mathcal{A},P_0,P_1}(\lambda)$. In addition, for all $i \in [0, 2^n - 1]$, the advantage of $\mathcal{A}$ in distinguishing between $\mathsf{Hyb}'_{\mathsf{i}+1}$ and $\mathsf{Hyb}'_{\mathsf{i}}$ can be at most $\mathsf{PosPIO}\ \mathsf{Adv}_{\mathcal{A},P_1,P_1}(\lambda)$. It follows that the advantage in distinguishing between Hyb'_{2^n} and Hyb'_0 is at most $2^n \cdot \mathsf{PosPIO}\ \mathsf{Adv}_{\mathcal{A},P_1,P_1}(\lambda)$.
- We finally note that Hyb'_0 is defined to be an obfuscation of P_1 as $i\mathcal{O}(1^\lambda, P_1)$.

Using the above hybrids we can draw a sequence that connects between an obfuscation. The sequence is $\mathsf{Hyb}_{\mathrm{start}}$, $\mathsf{Hyb}_0, \mathsf{Hyb}_1 \ldots \mathsf{Hyb}_{2^n}, \mathsf{Hyb}'_{2^n}, \mathsf{Hyb}'_{2^n-1}, \ldots, \mathsf{Hyb}'_0$. The theorem follows from the observations about the adversarial advantages between each hybrid.

■

Required Security from Positional Indistinguishability Obfuscation.: If all of the terms in our reduction were polynominal in n (and thus λ) then the only requirement we would have is that any poly-time algorithm have negligible advantage in each of our security games. However, there is an exponential term of 2^n attached to the positional game. Therefore we will need to use complexity leveraging and for all poly-time algorithms $\mathcal{A}$ and all $P_0, P_1 \in P_\lambda$ we demand that:

$$\mathsf{PosPIO}\ \mathsf{Adv}_{\mathcal{A},P_0,P_1}(\lambda) = \mathrm{negl}(\lambda) \cdot 2^{-n}$$

where $\mathrm{negl}(\lambda)$ is some negligible function. This requirement will be passed down to our next level of abstraction and eventually to our multi-linear encoding instantiation. At the instantiation level the security parameter will be increased to match this condition.

V. $ia\mathcal{O} \implies$ Positional Indistinguishability Obfuscation

We now describe how to build a positional indistinguishability obfuscation scheme from an input-activated obfuscation scheme.

To encode the position t, we will use two input-activated matrices, M_L^{t-1} and M_R^t. The matrix M_L^{t-1} will "activate" on inputs $x > t - 1$, meaning that for an input $x \geq t$, at least one column of M_L^{t-1} will evaluate to 1 on input x. For inputs $x < t$, all columns of M_L^{t-1} will evaluate to 0. M_R^t will have the complementary property that it is activated for inputs $x < t$ and not for inputs $x \geq t$. We will refer to M_L^{t-1} as a "left encoding" of the position t and to M_R^t as a "right" encoding of the position t.

To form a right encoding M_R^t, we can use the same encoding procedure used in [GLW14]: We consider the position t as a binary string $t = (t_1, t_2, \ldots, t_n) \in \{0,1\}^n$. We define an $n \times n \times 2$ input-activated matrix M_R^t by specifying how to fill in the j^{th} column for each $j \in [n]$:

To Set Column j::

- For $i < j$,
  $(M_R^t)_{i,j,0} = 1$,
  $(M_R^t)_{i,j,1} = \begin{cases} 0, & \text{if } t_i = 0; \\ 1, & \text{if } t_i = 1. \end{cases}$
- For $i = j$,
  $(M_R^t)_{i,j,0} = \begin{cases} 0, & \text{if } t_i = 0; \\ 1, & \text{if } t_i = 1. \end{cases}$
  $(M_R^t)_{i,j,1} = 0$
- For $i > j$,
  $(M_R^t)_{i,j,0} = 1 = (M_R^t)_{i,j,1}$.

We note some relevant properties of M_R^t. We observe that for every boolean string $y < t$, there is some column index j such that the associated boolean function evaluates to 1 on y, i.e. $f_j(y) = 1$. For every $y \geq t$, $f_j(y) = 0$ for all $j \in [n]$.

Here, we use "$<$" and "$\geq$" to denote the order induced by the usual ordering of integers, when we think of t, y as binary expansions with t_1, y_1 being the most significant bits. These observations are captured by the following lemmas, that are also in [GLW14], though we restate and prove them here for completeness.

Lemma V.1. *If $y < t$, then $f_j(y) = 1$ for some $j \in [n]$.*

Proof: Since $y < t$, there must be some index $j \in [n]$ such that $t_i = y_i$ for all $i < j$ and $t_j = 1$ while $y_j = 0$. We consider the j^{th} column of M_R^t. We claim that for all i, $(M_R^t)_{i,j,y_i} = 1$. To see this, we can consult our description of the j^{th} column of M_R^t above, noting that for $i < j$, whenever $y_i = 1$, then $t_i = 1$ as well (by definition of j). Thus, $f_j(y) = 1$. ■

Lemma V.2. *If $y \geq t$, then $f_j(y) = 0$ for all j.*

Proof: We let $k \in [n]$ denote an index such that $y_i = t_i$ for all $i \leq k$, and $y_{k+1} = 1$, $t_{k+1} = 0$, if $k + 1 \leq n$. For a column j where $j \leq k$, we observe that $(M_R^t)_{j,j,y_j} = 0$, since $y_j = t_j$. For any column j where $j > k$, we observe that $(M_R^t)_{k+1,j,y_{k+1}} = 0$. This is because $t_{k+1} = 0$ and $y_{k+1} = 1$. Hence, $f_j(y) = 0$ for all j. ■

This defines an effective right encoding of positions t from 0 to $2^n - 1$ (considering t as an integer). It will also be useful to have a right encoding of 2^n. For simplicity in our proof, we define the right encoding matrix of 2^n to be a small change from the right encoding of $2^n - 1$ that will ensure that the corresponding f will also evaluate to 1 on the position $2^n - 1$. Note that for $t = 2^n - 1$, only the diagonal entries of M_R^t are not completely filled with 1 slots. So we define $M_R^{2^n}$ to be the same as $M_R^{2^n-1}$, except that the first diagonal entry has both slots equal to 1.

We now construct a left encoding matrix M_L^t. This will also be a $n \times n \times 2$ input-activated matrix, but will be active on inputs $> t$.

To Set Column j::

- For $i < j$,
  $$(M_L^t)_{i,j,0} = \begin{cases} 1, & \text{if } t_i = 0; \\ 0, & \text{if } t_i = 1. \end{cases}$$
  $$(M_L^t)_{i,j,1} = 1,$$
- For $i = j$,
  $$(M_L^t)_{i,j,0} = 0$$
  $$(M_L^t)_{i,j,1} = \begin{cases} 1, & \text{if } t_i = 0; \\ 0, & \text{if } t_i = 1. \end{cases}$$
- For $i > j$,
  $$(M_L^t)_{i,j,0} = 1 = (M_L^t)_{i,j,1}.$$

We now prove the relevant properties of M_L^t:

Lemma V.3. *If $y \leq t$, then $f_j(y) = 0$ for all $j \in [n]$.*

Proof: For columns j such that $t_j = 1$, it is clear that $f_j(y) = 0$ for all y, since both diagonal slots of column j are 0. For a column j such that $t_j = 0$, if $y_j = 0$ as well, we will have $f_j(y) = 0$ due to the diagonal slot. If $y_j = 1$, then since $y \leq t$, there must be some row index $i < j$ such that $t_i = 1$ and $y_i = 0$. This leads to $f_j(y) = 0$ due to the slot at row i. ■

Lemma V.4. *If $y > t$, then $f_j(y) = 1$ for some $j \in [n]$.*

Proof: There must be some index j such that $y_j = 1$ and $t_j = 0$, while for all $i < j$, $y_i = 1$ whenever $t_i = 1$. We then have $f_j(y) = 1$ for the corresponding column j. ■

This defines an effective left encoding of positions t from 0 to $2^n - 1$ (considering t as an integer). It will also be useful to have a left encoding of -1. For simplicity in our proof, we define the left encoding matrix of -1 to be a small change from the left encoding of 0 that will ensure that some corresponding f_j will evaluate to 1 for every $y \geq 0$. Note that for $t = 0$, only the diagonal entries of M_L^t are not completely filled with 1 slots. So we define M_L^{-1} to be the same as M_L^0, except that the first diagonal entry has both slots equal to 1.

A. Construction

With these definitions in place, we are now prepared to present our construction.

Obfuscate$_{\mathrm{PIO}}(1^\lambda, C_0, C_1, t)$*::* This algorithm first forms a $n \times (2n+1) \times 2$ input-activated matrix by forming M_L^{t-1} as above, a $n \times 2$ "scratch column" S containing all 0 entries, and M_R^t as above. It concatenates these as $M := M_L^{t-1}|S|M_R^t$. It creates an ordered list of programs as $P = (P_1, \ldots, P_{2n+1})$ where $P_i = C_0$ for all $i \leq n+1$, and $P_i = C_1$ for all $i > n+1$. (This means C_0 will be associated with the columns of M_L^{t-1} and S, while C_1 will be associated with the columns of M_R^t.) It then calls $Create(1^\lambda, M, P)$ to form an input-activated obfuscation, and outputs the resulting object T.

Eval$_{\mathrm{PIO}}(T, x)$*::* This algorithm simply runs $Eval(T, x)$, the evaluation algorithm for the input-activated obfuscation.

Correctness.: For any security parameter λ, any pair of programs $C_0, C_1 \in \{\mathcal{C}_\lambda\}$, and for any input $x \in \{0,1\}^n$ we have that *Eval*$_{\mathrm{PIO}}\big($*Obfuscate*$_{\mathrm{PIO}}(1^\lambda, C_0, C_1, t), x\big) = Eval(Create(1^\lambda, M, P))$. If $x \geq t$, then M_L^{t-1} will have at least one column evaluating to 1 on x, but M_R^t will not. Hence $Eval(Create(1^\lambda, M, P)) = C_0(x)$ follows from the correctness of the input-activated obfuscation scheme. Similarly, it $x < t$, then M_R^t will have at least one column evaluating to 1 on x and M_L^{t-1} will not. In this case, $Eval(Create(1^\lambda, M, P)) = C_1(x)$ follows from the correctness of the input-activated obfuscation scheme.

B. Security

We first prove position indistinguishability.

Theorem V.5. *Position Indistinguishability for our Positional Obfuscation Scheme in Section V-A follows from inter-column security, intra-column security, completely inactive program security, and single-input program switching security of the underlying input-activated obfuscation scheme.*

Our proof of theorem V.5 will proceed as a hybrid argument, gradually changing the position encoding matrices to accommodate an increment of t. At the highest level, we organize our hybrid proof into five phases. We begin with the underlying input-activated matrix $M = M_L^{t-1}|S|M_R^t$ and program list $P = (C_0, \ldots, C_0, C_1, \ldots, C_1)$, where the first $n+1$ programs are C_0 and the remaining n programs are C_1. Recall here that the scratch column contains all 0's. We let Game$_0$ denote this setting, in which the attacker is given a positional obfuscation scheme derived from the input-activated obfuscation scheme for this matrix and programs.

We next define Game$_1$. In this game, the attacker will be given an input-activated obfuscation scheme for an adjusted input-activated matrix, though the list of associated programs remains the same as in Game$_0$. The new input-activated matrix will be $M_L^t|S_t|M_R^t$, where S_t is a column with entries $S_{i,t_i} = 1$ and $S_{i,1-t_i} = 0$ for all i. Note that this new scratch column S_t will be activated (i.e. have f_{n+1} evaluate to 1) if and only if the input is equal to t.

We next define Game$_2$. In this game, the underlying input-activated matrix is the same as in Game$_1$, but the affiliated program P_{n+1} for the scratch column is now C_1 instead of C_0. Crucially, transitioning to Game$_2$ will rely on the fact that $C_0(t) = C_1(t)$. We then define Game$_3$, in which the list of programs is the same as Game$_2$, but the underlying input-activated matrix is now $M_L^t|S|M_R^{t+1}$, where S is once again filled with all 0's (though still affiliated with C_1). We finally define Game$_4$, where the underlying input-activated matrix is as in Game$_3$, but the associated list of programs has reverted to $P_i = C_0$ for all $i \leq n+1$ and $P_i = C_1$ for all $i > n+1$. (In other words, the program associated to the scratch column is back to being C_0.)

For each adjacent pair of games, we will prove that the security properties of the underlying input-activated obfuscation scheme imply that no PPT attacker can distinguish between the two games with non-negligible advantage. Since Game$_0$ corresponds to a proper distribution for position t and Game$_4$ corresponds to a proper distribution for position $t+1$, we observe that these proofs in combination imply Theorem V.5. The proofs of Lemmas V.6 and V.8 below are very similar in spirit (and often in detail) to the proofs in [GLW14], so we defer them to the full version of this paper. Here we present the proofs of Lemmas V.7 and V.9, which are of a new flavor.

Lemma V.6. *If the input-activated obfuscation scheme has inter-column and intra-column security, then any PPT attacker can attain only a negligible advantage in distinguishing Game$_0$ from Game$_1$.*

Lemma V.7. *If the input-activated obfuscation scheme has single-input program switching security, then any PPT attacker can attain only a negligible advantage in distinguishing between Game$_1$ and Game$_2$.*

Proof: We suppose we have a PPT attacker $\mathcal{A}$ that can distinguish between Game$_1$ and Game$_2$ with non-negligible advantage. We will use this to create a PPT attacker $\mathcal{B}$ that achieves non-negligible advantage in the single-input program switching security game.

We employ the single-input program switching security game with input-activated matrix $M = M_L^t|S_t|M_R^t$, where S_t is the column corresponding to the characteristic function of t. We have the list of programs $(P_1, \ldots, P_{2n+1})$ with $P_i = C_0$ for $i \leq n+1$ and $P_i = C_1$ for $i > n+1$. The value of j will be $n+1$, and the alternate program P^* will be C_1. Note that $C_0(t) = C_1(t)$, so we have fulfilled all the requirements to apply the single-input program switching game.

$\mathcal{B}$ is given an input-activated obfuscation scheme T, and it must guess whether the alternate program was used. It passes T to $\mathcal{A}$ as the positional obfuscation scheme. If the original program list was used, this is properly distributed for Game$_1$. If the alternate program was used, this is properly distributed for Game$_2$. Hence $\mathcal{B}$ can leverage $\mathcal{A}$'s ability to distinguish these games to obtain a non-negligible advantage in the single-input program switching security game. ■

Lemma V.8. *If the input-activated obfuscation scheme has inter-column and intra-column security, then any PPT attacker can attain only a negligible advantage in distinguishing Game$_2$ from Game$_3$.*

Lemma V.9. *If the input-activated obfuscation scheme has completely inactive program security, then any PPT attacker can attain only a negligible advantage in distinguishing Game$_3$ from Game$_4$.*

Proof: We suppose we have a PPT attacker $\mathcal{A}$ that can distinguish between Game$_3$ and Game$_4$ with non-negligible advantage. We will use this to create a PPT attacker $\mathcal{B}$ that achieves non-negligible advantage in the completely inactive program security game.

We employ the completely inactive program security game with input-activated matrix $M_L^t|S|M_R^{t+1}$, where S is a column of all 0s. We have the list of programs $(P_1, \ldots, P_{2n+1})$ with $P_i = C_0$ for $i \leq n$ and $P_i = C_1$ for $i \geq n+1$. The value of j will be $n+1$, and the alternate program P^* will be C_0.

$\mathcal{B}$ is given an input-activated obfuscation scheme T, and it must guess whether the alternate program was used. It passes T to $\mathcal{A}$ as the positional obfuscation scheme. If the original program list was used, this is properly distributed for Game$_3$. If the alternate program was used, this is properly distributed for Game$_4$. Hence $\mathcal{B}$ can leverage $\mathcal{A}$'s ability to distinguish these games to obtain a non-negligible advantage in the completely inactive program security game. ■

We next prove Program Description Hiding A Security.

Theorem V.10. *Program Description Hiding A Security for our Positional Obfuscation Scheme in section V-A follows from intra-column security and completely inactive program security of the underlying input-activated obfuscation scheme.*

We begin with the underlying input-activated matrix $M = M_L^{-1}|S|M_R^0$ and program list $P = (C_0, \ldots, C_0, C_1, \ldots, C_1)$, where the first $n+1$ programs are C_0 and the remaining n programs are C_1. We let Game$_0$ denote this setting, in which the attacker is given a positional obfuscation scheme derived from the input-activated obfuscation scheme for this matrix and programs. For each z from 1 to n, we define Game$_z$ to be a game where the input-activated matrix is the same, but the first $n+1+z$ programs are C_0 and the remaining ones are C_1. Note that when we reach Game$_n$, C_1 no longer appears.

Theorem V.10 then follows from the following lemma:

Lemma V.11. *If intra-column and completely inactive program security hold for the underlying input-activated obfuscation scheme, then any PPT attacker has only a negligible advantage in distinguishing Game$_z$ from Game$_{z-1}$, for each z from 1 to n.*

Proof: The transition from Game$_{z-1}$ to Game$_z$ can be accomplished in three steps. First, we observe that column z of M_R^0 has 0s in both of its slot on row z. Thus we can invoke the intra-column security property to change this column to all 0s in all other rows as well. We can then invoke completely inactive program security to change the affiliated program from C_1 to C_0. Finally we can invoke intra-column security again to reset the other rows of this column to their proper values in M_R^0. ■

We last prove Program Description Hiding B Security with a similar argument.

Theorem V.12. *Program Description Hiding B Security for our Positional Obfuscation Scheme in section V-A follows from intra-column security and completely inactive program security of the underlying input-activated obfuscation scheme.*

We begin with the underlying input-activated matrix $M = M_L^{2^n-1}|S|M_R^{2^n}$ and program list $P = (C_0, \ldots, C_0, C_1, \ldots, C_1)$, where the first $n+1$ programs are C_0 and the remaining n programs are C_1. We let Game$_0$ denote this setting, in which the attacker is given a positional obfuscation scheme derived from the input-activated obfuscation scheme for this matrix and programs. For each z from 1 to $n+1$, we define Game$_z$ to be a game where the input-activated matrix is the same, but the first z programs are also C_1. Note that when we reach Game$_{n+1}$, C_0 no longer appears.

Theorem V.12 then follows from the following lemma:

Lemma V.13. *If intra-column and completely inactive program security hold for the underlying input-activated obfuscation scheme, then any PPT attacker has only a negligible advantage in distinguishing Game$_z$ from Game$_{z-1}$, for each z from 1 to $n+1$.*

Proof: The transition from Game$_{z-1}$ to Game$_z$ can be accomplished in three steps. First, we observe that for $z \leq n$, column z of $M_L^{2^n-1}$ has 0s in both of its slot on row z. Thus we can invoke the intra-column security property to change this column to all 0s in all other rows as well. We can then invoke completely inactive program security to change the affiliated program from C_0 to C_1. Finally we can invoke intra-column security again to reset the other rows of this column to their proper values in $M_L^{2^n-1}$. For $z = n + 1$, we can just apply completely inactive program security to change from C_0 to C_1, as the scratch column contains all 0s. ■

REFERENCES

[AJ15] Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. In *CRYPTO*, pages 308–326, 2015.

[BGK+13] Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Protecting obfuscation against algebraic attacks. In *EUROCRYPT*, pages 221–238, 2014.

[BGN05] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-dnf formulas on ciphertexts. In *TCC*, pages 325–341, 2005.

[BR14] Zvika Brakerski and Guy N. Rothblum. Virtual black-box obfuscation for all circuits via generic graded encoding. In *TCC*, pages 1–25, 2014.

[BV15] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. Cryptology ePrint Archive, Report 2015/163, 2015.

[BWZ14] Dan Boneh, David J. Wu, and Joe Zimmerman. Immunizing multilinear maps against zeroizing attacks. Cryptology ePrint Archive, Report 2014/930, 2014.

[BZ14] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In *CRYPTO*, pages 480–499, 2014.

[CHL+15] Jung Hee Cheon, Kyoohyung Han, Changmin Lee, Hansol Ryu, and Damien Stehlé. Cryptanalysis of the multilinear map over the integers. In *EUROCRYPT*, pages 3–12, 2015.

[CLT13] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Practical multilinear maps over the integers. In *CRYPTO*, pages 476–493, 2013.

[CLT15] Jean-Sebastien Coron, Tancrede Lepoint, and Mehdi Tibouchi. New multilinear maps over the integers. In *CRYPTO*, pages 267–286, 2015. .

[GGH13a] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In *EUROCRYPT*, pages 1–17, 2013.

[GGH+13b] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, pages 40–49, 2013.

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

[GGHR14] Sanjam Garg, Craig Gentry, Shai Halevi, and Mariana Raykova. Two-round secure MPC from indistinguishability obfuscation. In *TCC*, pages 74–94, 2014.

[GGHZ14] Sanjam Garg, Craig Gentry, Shai Halevi, and Mark Zhandry. Fully secure functional encryption without obfuscation. Cryptology ePrint Archive, Report 2014/666, 2014.

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In *STOC*, pages 467–476, 2013.

[GHMS14] Craig Gentry, Shai Halevi, Hemanta K. Maji, and Amit Sahai. Zeroizing without zeroes: Cryptanalyzing multilinear maps without encodings of zero. Cryptology ePrint Archive, Report 2014/929, 2014.

[GLW14] Craig Gentry, Allison B. Lewko, and Brent Waters. Witness encryption from instance independent assumptions. In *CRYPTO*, pages 426–443, 2014.

[GM84] S. Goldwasser and S. Micali. Probabilistic encryption. *Jour. of Computer and System Science*, 28(2):270–299, 1984.

[Kil88] Joe Kilian. Founding cryptography on oblivious transfer. In *STOC*, pages 20–31, 1988.

[PST14] Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation from semantically-secure multilinear encodings. In *CRYPTO*, pages 500–517, 2014.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *STOC*, pages 475–484, 2014.