# Indistinguishability Obfuscation from the Multilinear Subgroup Elimination Assumption

Craig Gentry Allison Lewko Amit Sahai Brent Waters

April 30, 2014

**Abstract**

We revisit the question of constructing secure general-purpose indistinguishability obfuscation ($i\mathcal{O}$), with a security reduction based on explicit computational assumptions. Previous to our work, such reductions were only known to exist based on instance-*dependent* 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.* (ePrint 2013), 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 an instance-*independent* computational assumption over multilinear maps, namely, the Multilinear Subgroup Elimination Assumption. Our 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.

# 1 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 [SW13], functional encryption [GGH$^+$13b], deniable encryption [SW13], two round multi-party computation [GGHR13], non-interactive multi-party key exchange [BZ13], 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$^+$13b, BR13, BGK$^+$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 Assumption [BGN05] from the bilinear setting. An important feature of this assumption is that it is instance-*independent*, as we elaborate below.

To describe the challenges that must be overcome to prove $i\mathcal{O}$ security 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.

**The dangers of instance-*dependent* assumption families.** In all previous constructions of indistinguishability obfuscation based on concrete assumptions [GGH$^+$13b, PST13], the security of the construction is based on an *instance dependent* family of assumptions over multilinear maps. By this, we mean that the assumption needed to prove security can depend quite intricately on the underlying indistinguishability obfuscation scheme. In essence, such previous assumptions were really an exponential family of assumptions: In [GGH$^+$13b], this was stated explicitly, with the assumption specifically quantifying over every pair of equivalent circuits. In the more recent work

of [PST13], a "meta-assumption" is given that contains another exponential family of assumptions, that in particular includes a separate assumption for every pair of equivalent circuits.

The essential danger in exponential assumption families is that there may be a special instance of the assumption (for example corresponding to a particular contrived pair of circuits) that is "hidden" among the assumptions, for which the assumption turns out to be false, but this possibility is hidden by the statement of the assumption. This danger is even more pronounced in meta-assumptions (as in [PST13]), where the succinct statement of the assumption effectively hides the variety of assumptions implied by the meta-assumption. We elaborate more on this in our Related Work section below.

Furthermore, such assumptions can also (explicitly or implicitly) embed 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. This is quite explicitly done in [GGH$^+$13b], and done implicitly in [PST13] where the structure of the $i\mathcal{O}$ scheme is embedded within the invoked instance of the meta-assumption.

**Our work: $i\mathcal{O}$ security from an instance-*independent* assumption.** With this motivation, we set out to answer the following question:

*Can we prove the security of indistinguishability obfuscation from an instance-*independent *assumption?*

In fact, our result works with a natural instance-*independent* 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 circuits, or even circuits 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 5.3.

We prove security based on this assumption by means of a new reduction technique that yields significant insight into the nature of $i\mathcal{O}$. Critically, the new hybrid argument underlying our assumption 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 hybrid argument technique is the technical heart of our paper. We elaborate on our techniques in Section 1.1.

**The necessity of $2^n$ 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 inherent, let us suppose that we want to prove security of an $i\mathcal{O}$ scheme under an instance-independent assumption $\alpha$, 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 $\alpha$. 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 doesn't 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 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 $\mathrm{PRG}(x) = y^*$, where $y^* = \mathrm{PRG}(x^*)$ is a constant embedded in the programs and PRG is a pseudo-random generator[1]. If this condition holds, then $P_0(x)$ outputs 0, but $P_1(x)$ outputs 1. It 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 $P^*(x^*)$. 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, such a reduction without a $2^n$ security loss is implausible.

Note that in contrast, a reduction to a family of instance-*dependent* 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 [PST13], for a pair of non-functionally equivalent programs, the [PST13] 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 [PST13], 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

[1]Note that by the pseudorandomness property, given only $y^*$ it is computationally difficult to determine if $y^* \in \mathrm{Range}(PRG)$, and thus to determine if $P_0$ and $P_1$ are functionally equivalent.

how to emulate the key features of multilinear maps that we need using only LWE.

### 1.1 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$^+$13b, PST13], the assumption itself was invoked to handle this case: In [GGH$^+$13b], the $i\mathcal{O}$ property itself followed directly from the assumption; in [PST13], the assumption was invoked to argue that a "merged version" of $P_0$ and $P_1$ can switch from using $P_0$ to using $P_1$ to perform the underlying computation. In this way, previous work relied on an underlying assumption that was powerful enough to encapsulate the actual workings of the obfuscation mechanism – but this is very unsatisfying, and it is the central drawback that we seek to avoid.

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* 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 hurdles – most notably, we need the obfuscation scheme itself to be decomposable into $2^n$ variations that somehow isolate each individual input. This will involve a security loss of $2^n$, but we have already argued above that such a security loss seems inherent.

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[2]. 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

[2]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 $P_1$.

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[3].

Another primary motivation of this invariant is what it means 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.

The above discussion is intended to give some intuitive "flavor" of our techniques. To actually implement these ideas, we proceed to define a number of intermediate abstractions that we use to translate this intuition to rigorous constructions and proofs.

**Progress on the Witness Encryption Front.** The starting point of our exploration will be the recent work of Gentry, Lewko, and Waters [GLW14]. As we already noted, the assumption under which we are able to base security is the Multilinear Subgroup Elimination Assumption introduced in [GLW14]. However, the problem addressed in this earlier work was different – witness encryption. In a witness encryption system [GGSW13], an encryptor will encrypt a message $m$ to an instance $x$ of some **NP** language $L$. A user can decrypt if they have an $\ell$-bit witness $w \in \{0,1\}^\ell$ that $x \in L$. A witness encryption system is secure if for any instance $x \notin L$, it should be hard to distinguish an encryption of $m_0$ from an encryption of $m_1$. Although witness encryption does not deal with obfuscating programs, it proves quite instructive to think about witness encryption.

In particular, the task of designing a security reduction for a witness encryption system involves a similar challenge to the one described above for $i\mathcal{O}$, as a reduction to an instance independent problem should intuitively discover if $x$ is indeed not in $L$. To address this, [GLW14] introduces a high level abstraction called *positional witness encryption*, that is inspired by work on fully collusion-resilient traitor tracing and broadcast encryption [BSW06, BW06]. Positional witness encryption behaves as standard witness encryption with the exception that the encryption algorithm takes an additional position input $t \in [0, 2^\ell]$. The added semantics are that decryption will work as long as the witness $w$ (interpreting it as an $\ell$-bit integer) used for decrypting is greater or equal to the position $t$. A security property for positional witness encryption asks that for any instance $x$, no attacker should be able to distinguish an encryption to index $t$ from one to index $t+1$, as long a $t$ is not a valid witness that $x \in L$. In this way, one can build a sequence of hybrids across all the possible witnesses one-by-one when $x \notin L$, since in this case no witnesses are valid.

This suggests to us an analogue that we call *positional indistinguishability obfuscation*, where a pair of programs are obfuscated, but where one program is active for the first $t$ inputs, while the other is active for inputs greater than or equal to $t$. (We elaborate on this further momentarily.)

[3]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.

At a high level, the [GLW14] strategy uses a hybrid where the position $t$ is embedded in a compact cryptographic data structure. This data structure is used to "save" the work done in the hybrid steps for each witness candidate tested so far. This data structure in turn is implemented by multilinear encodings and proven under assumptions that depend only on the instance size. We draw from [GLW14] in developing our own compact cryptographic data structure and encoding techniques, although since we must deal with programs, our constructions will also depart in fundamental ways from [GLW14]. In particular, the data structure of [GLW14] uses a kind of "divide-and-conquer" technique to attack each clause of a CNF formula separately. Since our work deals with general programs that we do not know how divide in this way, such a strategy will not work for us, and we develop a different strategy for building and using a compact cryptographic data structure for obfuscation. A significant technical contribution of our work is to construct the "individual security components" of our cryptographic data structure for obfuscation in such a way that we are then able to "embed" these components into the Multilinear Subgroup Elimination Assumption to argue security.

**Positional Indistinguishability Obfuscation.** As we mentioned above, our outermost layer of abstraction inserts a positional parameter $t$ into a new variant of indistinguishability obfuscation. In our context, this parameter $t$ will control an interpolation of two programs, allowing us to invoke one program for inputs less than $t$ and another program for the remaining inputs. The precision of $t$ will allow us to argue about a single isolated input, which we can then translate into a proof of security for all inputs through a hybrid argument. More precisely, a positional indistinguishability obfuscation scheme takes in two programs, $P_0$ and $P_1$, and a position $t$ that partitions the possible inputs. It will produce an obfuscated program that evaluates to $P_0(x)$ for inputs $x \geq t$ and evaluates to $P_1(x)$ for inputs $x < t$.

There are three required security properties. The first requires that when $t = 0$, an obfuscation of $P_0$ and $P_1$ is indistinguishable from an obfuscation of $P_0$ and $P_0$. This makes sense because $P_1$ does not effect the evaluation in this case. Analogously, the second security property requires that when $t = 2^n$ (where $n$ represents the bit length of inputs), an obfuscation of $P_0$ and $P_1$ is indistinguishable from an obfuscation of $P_1$ and $P_1$. The most interesting security property, termed position indistinguishability, requires that an obfuscation of $P_0, P_1$ with position $t$ is indistinguishable from one with position $t + 1$ when $P_0(t) = P_1(t)$. These properties allow us to perform a simple hybrid argument to obtain standard $i\mathcal{O}$. Naturally, this hybrid iterates over all the $2^n$ possible inputs, so it incurs $2^n$ security loss and thus necessitates complexity leveraging.

**Input-Activated Obfuscation.** We construct positional indistinguishability obfuscation from a mid-layer abstraction that we call input-activated obfuscation. This primitive defines a mild notion of obfuscation for a data structure that consists of an $n \times \ell \times 2$ matrix $M$ with entries in $\{0, 1\}$ and an ordered set of $\ell$ programs $P_1, \ldots, P_\ell$ associated to the columns of $M$. Jointly, this matrix and these programs describe a function on $n$-bit inputs that behaves as follows. First, each column $j$ of $M$ defines a boolean function $f_j : \{0, 1\}^n \to \{0, 1\}$ that describes on which inputs the $j$th column is active. On each input $x$, the evaluation of the obfuscated structure must agree with the value of $P_j(x)$ for every index $j \in [\ell]$ such that $f_j(x) = 1$. (This imposes a constraint on the combinations of matrices and programs that are valid to consider. Correctness is simply not required on any input for which these constraints are violated.)

There are four required security properties for an input-activated obfuscation scheme. Two of

the security properties, inter-column and intra-column security, describe small, localized changes to the matrix $M$ that can be made indistinguishably when the relevant columns have equal associated programs and precise conditions are met by the matrix entries involved. The remaining two security properties describe conditions under which a single program $P_j$ in the list can be changed, namely when its associated column function $f_j$ never evaluates to 1 (we call this completely inactive program security), or when $f_j$ evaluates to 1 on a single input where old and new program agree (we call this single-input program switching security).

Of course, one could imagine generalizing these properties to encompass a larger class of changes to the data structure that do not affect the evaluation, but this would be a step in the wrong direction. The fact that we can articulate very precise, very localized properties that suffice for our purposes is exactly what allows us to ultimately instantiate our abstraction from an instance independent assumption. The key features of our abstraction here are the program switching properties.

**Using Input-Activated Obfuscation.** We then show how to build positional indistinguishability obfuscation from input-activated obfuscation. Recall that an instance of our positional indistinguishability obfuscation is described by two programs $P_0, P_1$ and a position $t$. These can be embedded into the data structure of an input-activated obfuscation scheme by using $t$ to determine the matrix entries and using $P_0, P_1$ to populate the list of associated programs. More precisely, we form our matrix as a concatenation of three pieces, which we call $M_L^{t-1}$, $M_R^t$, and a scratch column $S$. The columns of $M_L^{t-1}$ will all be associated to the program $P_0$, while the columns of $M_R^t$ will all be associated to $P_1$. Initially, the scratch column $S$ will be associated to $P_0$. The main idea is that the columns of $M_L^{t-1}$ will be "activated" only on the inputs $x \geq t$, meaning their associated boolean functions $f_j$ evaluate to 0 for all inputs $x < t$. Analogously, the columns of $M_R^t$ will be activated only on inputs $x < t$. This ensures that the evaluation rules of the input-activated obfuscation scheme match up with the desired behavior of the positional indistinguishability obfuscation scheme.

To prove the position indistinguishability security property for the resulting positional indistinguishability obfuscation scheme, we must employ the security properties of the input-activated obfuscation primitive in a hybrid fashion to gradually change the encodings $M_L^{t-1}, M_R^t$ to $M_L^t, M_R^{t+1}$. To get from $M_L^{t-1}$ to $M_L^t$, we need to "deactivate" the input $t$. The techniques from [GLW14] give us a way of doing this using the scratch column, essentially allowing us to change to $M_L^{t-1}$ while populating the scratch column with entries that will activate solely on the input $t$. These adjustments can be accomplished with our inter-column and intra-column security properties because the scratch column and all the columns of the left matrix are associated to the same program $P_0$.

Once we have isolated this input $t$ in the scratch column, we can use our single-input program switching property to change the program associated with the scratch column to $P_1$. This relies on the fact that $P_0(t) = P_1(t)$. We can then use the intra-column and inter-column security properties similarly within $M_R^t$ and the scratch column to obtain $M_R^{t+1}$. We can then restore the scratch column to its original state. To prove the remaining security properties for positional indistinguishability obfuscation (ensuring that a program that is never activated can be switched), we can simply invoke completely inactive program security.

Our framework thus far is agnostic with respect to the type of program descriptions employed. To instantiate our framework from concrete assumptions about multilinear encodings, we will use matrix branching programs. However, if one obtained a different low-level instantiation for a

different form of program description, all of our abstractions and security reductions described so far could be reused.

**An Instantiation in a Model of Composite Order Multilinear Groups.** To instantiate our framework, we construct an input-activated obfuscation scheme in the setting of symmetric, composite order multilinear groups. We then show how the security properties of our construction follow from the Multilinear Subgroup Elimination Assumption. In the Appendix, we also discuss how to execute our construction from the candidate multilinear encodings of [CLT13].

Our construction draws upon the candidate constructions of $i\mathcal{O}$ in [GGH$^+$13b, BR13, BGK$^+$13], and essentially runs several instances of those constructions in parallel in different subgroups. We are able to simplify some aspects of the construction by developing alternative techniques to prevent "input mixing" and "partial evaluation" attacks. In [GGH$^+$13b, BR13, BGK$^+$13], additional algebraic structure was inserted to thwart such attacks, but we observe that these can be defeated already by the subgroup structures we are employing.

We focus our work on instantiating our methods in the symmetric, composite order setting as we feel it is the most instructive setting for our techniques. The work of [GLW14] describes methods for conversion to asymmetric and prime order multilinear groups. We expect the same variations could be obtained analogously for our input-activated obfuscation scheme, since at the lowest level we obtain an identical assumption to [GLW14].

### 1.2 Related Work

To more precisely compare the assumptions used in prior work to build obfuscation, we give a more detailed description of instance dependent assumption families. In such a family, an attacker will be given a set of group elements $\vec{H}$ by a challenger and either a set $\vec{T_0}$ or $\vec{T_1}$ depending on the challenger's coin flip. The job of the attacker is to guess whether it was given the set $\vec{T_0}$ or $\vec{T_1}$ in addition to $\vec{H}$ with better than negligible advantage. We (informally) say that an assumption is instance dependent if the distributions of $\vec{T_0}, \vec{T_1}, \vec{H}$ can depend on knowledge of the specific programs $P_0, P_1$ of the $i\mathcal{O}$ security game. The GGHRSW family of assumptions in particular was such that $\vec{T_0}$ and $\vec{T_1}$ were simply respective obfuscations of the programs $P_0$ and $P_1$ and $\vec{H}$ was empty.

The original work of GGHRSW also, separately, gave a proof of security for their construction against a highly idealized class of "generic matrix" attacks. Brakerski and Rothblum [BR13] and Barak et. al. [BGK$^+$13] showed that variations of the GGHRSW construction were secure against all generic multilinear attacks.

The work of Pass, Seth, and Telang [PST13] gives a proof under an instance-*dependent* family of multilinear assumptions that is best characterized as a "meta-assumption." A meta-assumption is a succinct description of a class of assumptions, which essentially says that any assumption that satisfies some set of properties is a true assumption. At a high level, the meta-assumption made in [PST13] states that, for all vectors $\vec{H}$ of group elements (with certain entropy guarantees), if *no generic multilinear attack* can distinguish two constant-length tuples $\vec{T_0}$ and $\vec{T_1}$ with respect to this vector $\vec{H}$, then no polynomial-time adversary can distinguish encodings of $(\vec{T_0}, \vec{H})$ from encodings of $(\vec{T_1}, \vec{H})$. However, the validity of this meta-assumption depends heavily on what class of "generic multilinear attacks" are considered. If one only requires that no polynomial-time generic multilinear attack can distinguish, then the assumption is actually *false*, as observed by [PST13]: this fact is quite hidden, and in essence it follows because one can embed a clever pair of circuits based on

the counterexample of [BGI+01] and use this to defeat the assumption. However, if the "generic multilinear attacks" considered are more restricted, then no such counter-example is known, and this forms the basis of the assumption actually made in [PST13].

# 2 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 [4] will be an encoding of a computable function. The system consists of two algorithms:

> ***Obfuscation.*** The algorithm $Obfuscate_{\text{PIO}}(1^\lambda, P_0, P_1, t)$ takes as input a security parameter $1^\lambda$, 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 $Eval_{\text{PIO}}(P, x)$ 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 2.1** ((Perfect) Correctness of Positional Indistinguishability Obfuscation)**.** *For any security parameter $\lambda$, 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*

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

## 2.1 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

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

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

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

**Definition 2.2** (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 $\mathit{negl}(\cdot)$ such that for all program description pairs $P_0, P_1 \in \{\mathcal{P}_\lambda\}$ we have $\mathsf{A} : \mathsf{HidingPIO}\ \mathsf{Adv}_{\mathcal{A},P_0,P_1}(\lambda) \leq \mathit{negl}(\lambda)$.*

**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

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

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

**Definition 2.3** (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 $\mathit{negl}(\cdot)$ such that for all program description pairs $P_0, P_1 \in \{\mathcal{P}_\lambda\}$ we have $\mathsf{B} : \mathsf{HidingPIO}\ \mathsf{Adv}_{\mathcal{A},P_0,P_1}(\lambda) \leq \mathit{negl}(\lambda)$.*

**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

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

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

**Definition 2.4** (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 $\mathit{negl}(\cdot)$ 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 $\mathsf{PosPIO}\ \mathsf{Adv}_{\mathcal{A},P_0,P_1,t}(\lambda) \leq \mathit{negl}(\lambda)$.*

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

## 2.2 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 $x \in [0, 2^n - 1]$ : $P_0(x) = P_1(x)$. 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 $\mathit{Obfuscate}_{\mathrm{PIO}}(1^\lambda, P, P, t = 0)$.

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

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 2.5.** *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 $\mathcal{A}$*

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

*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:

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

  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 $\mathsf{Hyb}_{\text{start}}$ and $\mathsf{Hyb}_0$ can be at most $\mathsf{A} : \mathsf{HidingPIO\ Adv}_{\mathcal{A},P_0,P_1}(\lambda)$. In addition, for all $i \in [0, 2^n - 1]$, the advantage of $\mathcal{A}$ in distinguishing between $\mathsf{Hyb}_i$ and $\mathsf{Hyb}_{i+1}$ can be at most $\mathsf{PosPIO\ Adv}_{\mathcal{A},P_0,P_1}(\lambda)$. This follows from the fact that $P_0(i) = P_1(i)$. It follows that the advantage in distinguishing between $\mathsf{Hyb}_0$ and $\mathsf{Hyb}_{2^n}$ is at most $2^n \cdot \mathsf{PosPIO\ Adv}_{\mathcal{A},P_0,P_1}(\lambda)$.

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

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

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

- We finally note that $\mathsf{Hyb}'_0$ 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 $\mathsf{Hyb}_{\text{start}}, \mathsf{Hyb}_0, \mathsf{Hyb}_1 \ldots \mathsf{Hyb}_{2^n}, \mathsf{Hyb}'_{2^n}, \mathsf{Hyb}'_{2^n-1}, \ldots, \mathsf{Hyb}'_0$. 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 $\lambda$) 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:

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

where $\text{negl}(\lambda)$ 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.

# 3 Input-Activated Obfuscation

In this section, we will describe an abstraction that we call *Input-Activated Obfuscation* ($ia\mathcal{O}$). 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 $\ell$ programs, $P = (P_1, P_2, \ldots, P_\ell)$ from a specified family of programs $\{\mathcal{P}_\lambda\}$, parameterized by a security parameter $\lambda$. We abuse standard terminology slightly by saying that such a matrix $M$ has $n$ rows and $\ell$ 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 $\lambda$, an $n \times \ell \times 2$ input-activated matrix $M$, and an ordered set $P = (P_1, P_2, \ldots, P_\ell)$ of $\ell$ 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 $\lambda$, 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 3.1.** *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 $M, P, j, k, i^*, \beta$.*

**Intra-column Security Game.** This game is parameterized by a security parameter $\lambda$, 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 3.2.** *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 $\lambda$, 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 3.3.** *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 $\lambda$, 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 3.4.** *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* $M, P, P^*, j$*.*

# 4 $ia\mathcal{O} \implies$ 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 4.1.** *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 4.2.** *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 4.3.** *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 4.4.** *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.

### 4.1 Construction

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

***Obfuscate***$_{\text{PIO}}(1^\lambda, C_0, C_1, t)$**:** 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***$_{\text{PIO}}(T, x)$**:** This algorithm simply runs $Eval(T, x)$, the evaluation algorithm for the input-activated obfuscation.

**Correctness.** For any security parameter $\lambda$, any pair of programs $C_0, C_1 \in \{\mathcal{C}_\lambda\}$, and for any input $x \in \{0,1\}^n$ we have that $Eval_{\text{PIO}}\big(Obfuscate_{\text{PIO}}(1^\lambda, C_0, C_1, t), x\big) = Eval(Create(1^\lambda, M, P))$. 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.

### 4.2 Security

We first prove position indistinguishability.

**Theorem 4.5.** *Position Indistinguishability for our Positional Obfuscation Scheme in Section 4.1 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 4.5 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 $\text{Game}_0$ 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 $\text{Game}_1$. 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 $\text{Game}_0$. 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 $\text{Game}_2$. In this game, the underlying input-activated matrix is the same as in $\text{Game}_1$, but the affiliated program $P_{n+1}$ for the scratch column is now $C_1$ instead of $C_0$. Crucially, transitioning to $\text{Game}_2$ will rely on the fact that $C_0(t) = C_1(t)$. We then define $\text{Game}_3$, in which the list of programs is the same as $\text{Game}_2$, 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 $\text{Game}_4$, where the underlying input-activated matrix is as in $\text{Game}_3$, 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 $\text{Game}_0$ corresponds to a proper distribution for position $t$ and $\text{Game}_4$ corresponds to a proper distribution for position $t+1$, we observe that these proofs in combination imply Theorem 4.5. The proofs of Lemmas 4.6 and 4.8 below are very similar in spirit (and often in detail) to the proofs in [GLW14], so we defer them to appendices B and C. Here we present the proofs of Lemmas 4.7 and 4.9, which are of a new flavor.

**Lemma 4.6.** *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 $\text{Game}_0$ from $\text{Game}_1$.*

**Lemma 4.7.** *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 $\text{Game}_1$ and $\text{Game}_2$.*

*Proof.* We suppose we have a PPT attacker $\mathcal{A}$ that can distinguish between $\text{Game}_1$ and $\text{Game}_2$ 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 $\text{Game}_1$. If the alternate program was used, this is properly distributed for $\text{Game}_2$. 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 4.8.** *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 $\text{Game}_2$ from $\text{Game}_3$.*

**Lemma 4.9.** *If the input-activated obfuscation scheme has completely inactive program security, then any PPT attacker can attain only a negligible advantage in distinguishing $\text{Game}_3$ from $\text{Game}_4$.*

*Proof.* We suppose we have a PPT attacker $\mathcal{A}$ that can distinguish between $\text{Game}_3$ and $\text{Game}_4$ 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 $\text{Game}_3$. If the alternate program was used, this is

properly distributed for $\text{Game}_4$. 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 4.10.** *Program Description Hiding A Security for our Positional Obfuscation Scheme in section 4.1 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 $\text{Game}_0$ 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 $\text{Game}_z$ 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 $\text{Game}_n$, $C_1$ no longer appears.

Theorem 4.10 then follows from the following lemma:

**Lemma 4.11.** *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 $\text{Game}_z$ from $\text{Game}_{z-1}$, for each $z$ from 1 to $n$.*

*Proof.* The transition from $\text{Game}_{z-1}$ to $\text{Game}_z$ 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 4.12.** *Program Description Hiding B Security for our Positional Obfuscation Scheme in section 4.1 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 $\text{Game}_0$ 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 $\text{Game}_z$ 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 $\text{Game}_{n+1}$, $C_0$ no longer appears.

Theorem 4.12 then follows from the following lemma:

**Lemma 4.13.** *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 $\text{Game}_z$ from $\text{Game}_{z-1}$, for each $z$ from 1 to $n+1$.*

*Proof.* The transition from $\text{Game}_{z-1}$ to $\text{Game}_z$ 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. □

# 5 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 $5 \times 5$ 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.

### 5.1 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 $\lambda$, 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 $G$.

### 5.2 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:

**$\mathbf{Create}_{lite}(\lambda, M, P)$:** This algorithm takes in a security parameter $\lambda$, an $n \times \ell \times 2$ input-activated matrix $M$, and an ordered set $P = (P_1, \ldots, P_\ell)$ of programs from $P_\lambda$. We require that all $P_j$ are oblivious matrix branching programs of length $z$ for inputs of length $n$, represented as $z$ pairs of $5 \times 5$ 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 $\ell$, 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 $\gamma_i$ 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 $\gamma_i$ from the subgroup of order $p_i$.

For each $j$ from 1 to $\ell$, 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 $T$.

**$\mathbf{Eval}_{lite}(T, x)$:** 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 $5 \times 5$ collection of elements $\{g_{w,v,t,b}\}$, as $w, v$ range over $[5]$. We view $k_{t,b}$ as a $5 \times 5$ 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 $5 \times 5$ 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 $\text{Create}_{lite}$, $\text{Eval}_{lite}$ 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:

**$\text{Create}(\lambda, M, P)$:** This algorithm takes in a security parameter $\lambda$, an $n \times \ell \times 2$ input-activated matrix $M$, and an ordered set $P = (P_1, \ldots, P_\ell)$ of programs from $P_\lambda$. 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 $T$.

**$\text{Eval}(T, x)$:** 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 $\text{Create}_{lite}$ and $\text{Eval}_{lite}$, as the additional columns are not activated on any inputs.

### 5.3 The Multilinear Subgroup Elimination Assumption

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

**The $(\mu, \nu)$-multilinear subgroup elimination assumption** This assumption is parameterized by positive integers $\mu$, and $\nu$. It concerns a $\mu$-linear group of order $N = a_1 \ldots a_\mu b_1 \ldots b_\nu c$, where $a_1, \ldots, a_\mu, b_1, \ldots, b_\nu, c$ are $\mu + \nu + 1$ distinct primes. We give out generators $g_{a_1}, \ldots, g_{a_\mu}, g_{b_1}, \ldots, g_{b_\nu}$ 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 $T$.

### 5.4 Security Properties

**Lemma 5.1.** *For $\mu := z + n$ and $\nu := \ell + 2z - 1$, the $(\mu, \nu)$-multilinear subgroup elimination assumption implies inter-column security for our algorithms $\text{Create}_{lite}$, $\text{Eval}_{lite}$ in Section 5.2.*

*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 $M, P, j, k, i, \beta$. We will create a PPT attacker $\mathcal{B}$ that achieves a non-negligible advantage in breaking the $(\mu, \nu)$-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 $a_1 \cdots a_\mu$.

$\mathcal{B}$ 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 $b_1, \ldots, b_\nu$. Then $\mathcal{B}$ samples the randomized matrices $B_{j,t,b}$ for all $j \in [\ell]$, $t \in [z]$, and $b \in \{0, 1\}$.

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 $\gamma_1$, ..., $\gamma_{t-1}$, $\gamma_{t+1}$, ..., $\gamma_{z+n}$, $u_{j'}$ 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 $\gamma_1$, ..., $\gamma_{z+y-1}$, $\gamma_{z+y+1}$, ..., $\gamma_{z+n}$ and $d_{s,b'}$ elements appropriately by using the given subgroup generators. It can similarly sample elements $u_{j'}$ for $j' \neq j, k$. 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 ($\alpha$ mod $c$) and ($\alpha$ 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 $h_{z+i}^{\alpha}$ 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 5.2.** *[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 5.3.** *For $\mu := z + n$ and $\nu := \ell + 2z - 1$, the $(\mu, \nu)$-multilinear subgroup elimination assumption implies single-input program switching security for our algorithms $\mathsf{Create}_{lite}$, $\mathsf{Eval}_{lite}$ in Section 5.2.*

*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. $\mathrm{Exp}_0$ 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 $\mathrm{Exp}_f$ to be like $\mathrm{Exp}_0$, 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 $\mathrm{Exp}_z$, only one matrix will appear in the $q_j$ subgroup per slot.)

We first argue that for each such $f$, $\mathrm{Exp}_f$ is indistinguishable from $\mathrm{Exp}_{f-1}$, 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 $\mathrm{Exp}_f$ from $\mathrm{Exp}_{f-1}$. 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

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

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 $\text{Exp}_{f-1}$. If not, it has properly simulated $\text{Exp}_f$. This allows us to argue that under this multilinear subgroup elimination assumption, $\text{Exp}_0$ is computationally indistinguishable from $\text{Exp}_z$.

Next, we argue that the distribution of $\text{Exp}_z$ is statistically close to the distribution of an $\text{Exp}_z^*$ 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 $\text{Exp}_z$ to $\text{Exp}_z^*$ 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 $\text{Exp}_z$ is only a negligible statistical distance from the distribution required for Lemma 5.2, 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 $\text{Exp}_z^*$ is negligibly close to the distribution of $\text{Exp}_z$.

Once we are at $\text{Exp}_z^*$, 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 5.4.** *For $\mu := z + n$ and $\nu := \ell + 2z - 1$, the $(\mu, \nu)$-multilinear subgroup elimination assumption implies completely inactive program security for algorithms $\text{Create}_{lite}$, $\text{Eval}_{lite}$ in Section 5.2.*

*Proof.* This follows from the same hybrid argument used in the proof of Lemma 5.3, 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 5.5.** *For $\mu := z + n$ and $\nu := \ell + n\ell + 2z - 1$, the $(\mu, \nu)$-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 5.2.*

*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 5.1, 5.3, and 5.4, as the columns of the matrix $M'$ that is input to $\text{Create}_{lite}$ 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 $\text{Create}_{lite}$, $\text{Eval}_{lite}$. 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 $C$. □

# 6 Implementing Our Scheme with Graded Encodings

We instantiate our $i\mathcal{O}$ scheme with an extension of the graded encoding scheme described by Coron, Lepoint and Tibouchi (CLT) [CLT13]. The adapted encodings are the same as in [GLW14], and indeed we re-use their $(\mu, \nu)$-multilinear subgroup elimination assumption over these encodings.

We very briefly review some of the graded encoding procedures. (See Appendix D for a thorough review of these procedures, along with other aspects of the extended CLT encodings described in [GLW14].) $\mathsf{InstGen}(\lambda, \kappa, \sigma)$ takes as input the security parameter $\lambda$, the multilinearity parameter $\kappa$, and a ring dimension parameter $\sigma$, and generates parameters $\mathsf{params}$ for the graded encoding system. These parameters include a composite modulus $Q = \prod_{i\in[\sigma],\theta\in[\Theta]} Q_{i,\theta}$, a product of "big" primes, and a zero test parameter $p_{zt}$. They also include a composite modulus $N = \prod_{i\in[\sigma],\theta\in[\Theta]} p_{i,\theta}$, a product of "small" primes. The actual encoded values will be modulo $N$, whereas addition and multiplication of encodings will be modulo $Q$. The value $\sigma$ corresponds to the number of prime divisors needed in our composite-order group $i\mathcal{O}$ scheme. Each of the $\sigma$ primes in the original scheme is associated to $\Theta = \Theta(\lambda, \kappa)$ primes in the CLT moduli. Our $i\mathcal{O}$ scheme's Create algorithm will use the procedure $\mathsf{SubRGen}(\mathsf{esk}, i, S)$, a private randomized algorithm that takes as input a secret encoding parameter $\mathsf{esk}$, a parameter $i$ indicating an encoding level, and a subset $S \subset [\sigma]$. This procedure then generates a number modulo $m \in \mathbb{Z}_N$ that is random modulo $p_{i,\theta}$ for all $i \in S$ and is 0 modulo $p_{i,\theta}$ for all $i \in [\sigma] \setminus S$. Finally, it outputs a level-$i$ encoding $\mathcal{E}_i^{(m)}$ of $m$. (We will only need it to output level-1 encodings, and please see the Appendix for details on how encoding works.) Create will also use the $\mathsf{ReRand}(\mathsf{params}, u)$ procedure, which works by adding a random encoding in $\mathcal{E}_1^{(0)}$ to $u \in \mathcal{E}_1^{(m)}$ to generate a new statistically random encoding $u' \in \mathcal{E}_1^{(m)}$. Our $i\mathcal{O}$ scheme's Eval algorithm will use the addition and multiplication procedures $+$ and $\times$, as well as the zero test $\mathsf{IsZero}(\mathsf{params}, p_{zt}, u)$ that uses the zero-tester $p_{zt}$ to distinguish whether $u$ encodes 0 at level $\kappa$.

There are some minor differences with extended CLT encodings as described in [GLW14], since the GLW setting was somewhat simpler. In GLW, decryption works by a simple application of the multilinear map that directly maps level-1 encodes to level-$\kappa$ without really using any intermediate levels. By contrast, as we multiply matrices (which requires additions and multiplications of individual entries) we need to exploit the additive structure of intermediate groups – i.e., we need a true graded encoding. Another difference is that GLW only needed to encode values that were either 0 or random modulo various primes. In contrast, we need to encode *particular* values – e.g., we need to ensure that the particular Kilian randomizing matrices are used consistently in our randomized branching programs. Thus, while $N$ was not explicitly included in the GLW parameters since it was not needed, we include it in $\mathsf{params}$ in our system. (Including it does not appear to lead to attacks, and of course the prime factors of $N$ and $Q$ remain secret.) Also, we include a level-1 encoding $c^*$ of 1 in $\mathsf{params}$.

Now, we describe the "lite" version of our $i\mathcal{O}$ scheme in terms of the extended CLT graded encoding system. (The "lite" to "final" transformation is straightforward and analogous to that in Section 5.)

**$\mathbf{Create}_{lite}(\lambda, M, P)$:** This algorithm takes in a security parameter $\lambda$, an $n \times \ell \times 2$ input activated matrix $M$, and an ordered set $P = (P_1, \ldots, P_\ell)$ of programs from $P_\lambda$ of length $z$ represented by $5 \times 5$ matrices $\{A_{j,t,b} : j \in [\ell], t \in [z], b \in \{0,1\}\}$. It then calls $(\mathsf{params}, p_{zt}) \leftarrow \mathsf{InstGen}(\lambda, z+n, 3z+n+\ell)$ to produce a $(z+n)$-graded encoding system for $\sigma = 3z + n + \ell$. To mirror the description our $i\mathcal{O}$ scheme in Section 5, the CLT modulus is

$$N = \prod_{\theta \in [\Theta]} q_{1,\theta} \cdots q_{\ell,\theta} \cdot p_{1,\theta} \cdots p_{z+n,\theta} \cdot \prod_{s=1}^{z} r_{s,0,\theta} \cdot r_{s,1,\theta}.$$

For convenience, we let $\mathcal{Q} \setminus \{i\}$ denote the set of indices corresponding to the $q$ primes, except $q_{i,1}, \ldots, q_{i,\theta}$, and use similar notation $\mathcal{P} \setminus \{i\}$ and $\mathcal{R} \setminus \{i\}$ for the $p$ primes and $r$ primes.

To generate the "program carrying encodings" for our $i\mathcal{O}$ scheme – that is, encodings of the entries in the $5 \times 5$ matrices of the branching program used in our scheme – we proceed as follows. First, we choose uniformly random matrices $R_1, \ldots, R_{z-1}$ in $\mathbb{Z}_N^{5\times 5}$, and let $R_0, R_z$ be the identity matrix. For each $t \in [z]$ and $b \in \{0,1\}$, we define $B_{t,b}$ to be the matrix equal to $R_{t-1}^{-1} A_{j,t,b} R_t$ modulo $\prod_{\theta \in [\Theta]} p_{j,\theta}$ for $j \in [\ell]$, but equal to 0 modulo the other primes. We compute an individual encoding $c_{w,v,t,b}$ – with $w, v \in [5]$, $t \in [z]$, $b \in \{0,1\}$ – as follows. Set $c_0 \leftarrow \mathsf{SubRGen}(\mathsf{esk}, 1, \mathcal{P} \setminus \{t\})$, set $c_1 \leftarrow \mathsf{SubRGen}(\mathsf{esk}, 1, \mathcal{R} \setminus \{(t, 1-b)\})$, set $c_2 \leftarrow c^* \times B_{t,b}[w, v]$, and finally output $c_{w,v,t,b} \leftarrow \mathsf{ReRand}(\mathsf{params}, c_0 + c_1 + c_2)$.

To generate an enforcing encoding $c_{y,b}$ for $y \in [n]$ and $b \in \{0,1\}$, set $c_0 \leftarrow \mathsf{SubRGen}(\mathsf{esk}, 1, \mathcal{P} \setminus \{z+y\})$, set $c_1 \leftarrow \mathsf{SubRGen}(\mathsf{esk}, 1, \mathcal{R} \setminus \{(s, b') : \alpha(s) = y \wedge b' = b\})$, set $c_2 \leftarrow \mathsf{SubRGen}(\mathsf{esk}, 1, \mathcal{Q} \setminus \{j : M_{y,j,b} = 0\})$, and finally output $c_{y,b} \leftarrow \mathsf{ReRand}(\mathsf{params}, c_0 + c_1 + c_2)$.

The encodings $\{c_{w,v,t,b}\}$, $\{c_{y,b}\}$, with $\mathsf{params}$, constitute the input-activated obfuscation $T$.

**$\mathbf{Eval}(T, x)$:** On input $x \in \{0,1\}^n$, we evaluate $T$ as follows. We compute:

$$C_{1,x_{\alpha(1)}} \times \cdots \times C_{z,x_{\alpha(z)}},$$

where $C_{t,b}$ is the matrix of encodings $\{c_{w,v,t,b}\}$, and matrix multiplication is computed over the encodings using $+$ and $\times$ in the natural way.

We will then choose a single encoding off the diagonal. 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 encoding $u$. We then compute:

$$e(u, c_{1,x_1}, \ldots, c_{n,x_n}). \tag{3}$$

We apply the zero test to the result of (3). If the zero test detects a zero, we output 1. Otherwise, we output 0.

**Correctness and Security** The correctness analysis here is not substantially different from that in Section 5, aside from the fact that the noise level and parameters need to be set to ensure correctness. These issues are discussed in Appendix D.

The $(\mu, \nu)$-multilinear subgroup elimination assumption over extended CLT encodings is as follows. (The assumption refers to a procedure $\mathsf{SubRSamp}$ described in Appendix D.)

**$(\mu, \nu)$-multilinear subgroup elimination assumption for graded encodings** The challenger runs $(\mathsf{params}, p_{zt}) \leftarrow \mathsf{InstGen}(\lambda, \mu + \nu + 1)$ to obtain a graded encoding system. For $S$ equal to $\{1\}, \ldots, \{\mu + \nu\}$, and $\{\mu + \nu + 1\} \cup [\mu] \setminus \{1\}, \ldots, \{\mu + \nu + 1\} \cup [\mu] \setminus \{\mu\}$, it populates a set of encodings $\mathcal{G}_S$ by running the procedure $\mathsf{SubRGen}(\mathsf{esk}, 1, S)$ enough times to obtain a "generating set" (that could later be used in $\mathsf{SubRSamp}$). The challenge term $u^*$ is an encoding that is the output of $\mathsf{SubRGen}(\mathsf{esk}, 1, S^*)$, where $S^*$ equals either $\{\mu + \nu + 1\} \cup [\mu]$ or $[\mu]$. The task is, given the $\mathcal{G}_S$'s and $u^*$, to distinguish which distribution $u^*$ comes from.

This is directly analogous to the assumption described in Section 5, with the minor difference that, in the context of groups, the generating set is a single group element, whereas we cannot assume that this works in general for graded encoding systems.

The security reductions to the $(\mu, \nu)$-multilinear subgroup elimination assumption for graded encodings are directly analogous to the proofs given in Section 5, with the difference that the $\mathsf{SubRSamp}$ and $\mathsf{ReRand}$ procedures are needed to ensure that certain distributions are statistically identical.

## References


[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *IACR Cryptology ePrint Archive*, 2001:69, 2001.

[BGK+13] Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Protecting obfuscation against algebraic attacks. Cryptology ePrint Archive, Report 2013/631, 2013. `http://eprint.iacr.org/`.

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

[BR13] Zvika Brakerski and Guy N. Rothblum. Virtual black-box obfuscation for all circuits via generic graded encoding. Cryptology ePrint Archive, Report 2013/563, 2013. `http://eprint.iacr.org/`.

[BSW06] Dan Boneh, Amit Sahai, and Brent Waters. Fully collusion resistant traitor tracing with short ciphertexts and private keys. In *EUROCRYPT*, 2006.

[BW06] Dan Boneh and Brent Waters. A fully collusion resistant broadcast, trace, and revoke system. In *ACM Conference on Computer and Communications Security*, pages 211–220, 2006.

[BZ13] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. Cryptology ePrint Archive, Report 2013/642, 2013. `http://eprint.iacr.org/`.

[CLT13] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Practical multilinear maps over the integers. In *CRYPTO (1)*, pages 476–493, 2013.

[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.

[GGHR13] Sanjam Garg, Craig Gentry, Shai Halevi, and Mariana Raykova. Two-round secure mpc from indistinguishability obfuscation. Cryptology ePrint Archive, Report 2013/601, 2013. `http://eprint.iacr.org/`.

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

[GLW14] Craig Gentry, Allison Bishop Lewko, and Brent Waters. Witness encryption from instance independent assumptions. Cryptology ePrint Archive, Report 2014/273, 2014. `http://eprint.iacr.org/`.

[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.

[PST13] Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation from semantically-secure multilinear encodings. Cryptology ePrint Archive, Report 2013/781, 2013. `http://eprint.iacr.org/`.

[SW13] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: Deniable encryption, and more. Cryptology ePrint Archive, Report 2013/454, 2013. `http://eprint.iacr.org/`.

# A Definition of Indistinguishability Obfuscation

**Definition A.1** (Indistinguishability Obfuscator ($i\mathcal{O}$))**.** *A pair of uniform PPT machines* $(i\mathcal{O}, Eval)$ *is called an* indistinguishability obfuscator *for a circuit class* $\{\mathcal{C}_\lambda\}$ *with inputs of size* $n(\lambda)$ *if the following conditions are satisfied:*

- *For all security parameters* $\lambda \in \mathbb{N}$*, for all* $C \in \mathcal{C}_\lambda$*, for all inputs* $x \in \{0,1\}^{n(\lambda)}$*, we have that*

$$\Pr\Big[Eval\big(i\mathcal{O}(\lambda, C), x\big) = C(x)\Big] = 1$$

- *There exists a negligible function* $\alpha$ *such that the following holds: For any (not necessarily uniform) PPT adversaries Samp, D, where* $Samp(1^\lambda)$ *outputs a tuple* $(C_0, C_1, \sigma)$*, where* $C_0, C_1 \in \mathcal{C}_\lambda$*, we have that:*

  *If for all* $\lambda$*, it is true that* $\Pr[\forall x \in \{0,1\}^{n(\lambda)}, C_0(x) = C_1(x) : (C_0, C_1, \sigma) \leftarrow Samp(1^\lambda)] > 1 - \alpha(\lambda)$*, then we have:*

$$\Big|\Pr\left[D(\sigma, i\mathcal{O}(\lambda, C_0)) = 1 : (C_0, C_1, \sigma) \leftarrow Samp(1^\lambda)\right] - \Pr\left[D(\sigma, i\mathcal{O}(\lambda, C_1)) = 1 : (C_0, C_1, \sigma) \leftarrow Samp(1^\lambda)\right]\Big| \leq \alpha(\lambda)$$

# B Proof of Lemma 4.6

For this game transition, the list of programs is fixed, and we are going from an underlying input-activated matrix of $M_L^{t-1}|S|M_R^t$ to a matrix of $M_L^t|S_t|M_R^t$. We will preform this transition gradually over several steps. These steps are all very similar to the hybrid proof for positional witness encryption in [GLW14].

We let $M_0 := M_L^{t-1}|S|M_R^t$ denote our starting input-activated matrix, and we recall $\text{Game}_0$ is the security game corresponding to this. We first consider $t > 0$. We then have that $0 \leq t - 1 < 2^n - 1$, so if we consider the bits of $t - 1$ in order from most significant to least significant, there is some bit index $r$ such that the $r^{th}$ bit is 0 and the remaining bits are all equal to 1. The bits of $t$ will then have a 1 in the $r^{th}$ position and 0s following. We observe by our definition of $M_L^t$ that is is only the bottom right submatrix of $M_L^{t-1}$ corresponding to columns and rows $\geq r$ that changes as we move from $M_L^{t-1}$ to $M_L^t$.

As a first step, we will transition to $M_1 := M_L^{t-1}|\tilde{S}_t|M_R^t$, where $\tilde{S}_t$ is a column that matches the $r^{th}$ column of $M_L^{t-1}$ (note that this column activates on $t$). We let $\text{Game}_{0.1}$ denote a variant of the security game using $M_1$ in place of $M_0$.

**Lemma B.1.** *If the input-activated obfuscation scheme has inter-column security, then any PPT attacker has only a negligible advantage in distinguishing* $Game_0$ *from* $Game_{0.1}$*.*

*Proof.* We can invoke the inter-column security game repeatedly, once for each entry of the scratch column that we to change from a 0 to a 1 in order to make it match the $r^{th}$ column of $M_L^{t-1}$. Throughout this process, that $r^{th}$ column "dominates" the scratch column, in that it contains a 1 in every slot where the scratch column contains a 1, thus all of our invocations satisfy the constraints of the inter-column security game. □

We next define $M_{2,z}$ for each $z$ from 0 to $n-r$ as follows. The scratch column and rightmost matrix $M_R^t$ will the same for each $M_{2,z}$ as in $M_1$. In $M_{2,0}$, the left matrix will differ from $M_L^{t-1}$ only in columns and rows $\geq r$. In these entries, all non-diagonal entries will match their specification in $M_L^t$, while all diagonal entries will continue to match their specifications in $M_L^{t-1}$. For $z = 1$ through $n-r$, the diagonal entries of $M_{2,z}$ for indices $r+1$ to $r+z$ will also match their specifications in $M_L^t$. This means that once we arrive at $M_{2,n-r}$, all but the $r^{th}$ column of the left matrix will match $M_L^t$. We let $\text{Game}_{0.2,z}$ denote the variant of the security game corresponding to the input-activated matrix $M_{2,z}$.

**Lemma B.2.** *If the input-activated obfuscation scheme has intra-column security, then any PPT attacker has only a negligible advantage in distinguishing $\text{Game}_{0.1}$ from $\text{Game}_{0.2,0}$.*

*Proof.* We observe that for each column $r+1$ to $n$ in $M_L^{t-1}$, the diagonal row has 0s in both slots. This allows us to apply the intra-column security property $n-r$ times in order to adjust the remaining rows of these columns to match their specifications in $M_L^t$. □

**Lemma B.3.** *If the input-activated obfuscation scheme has inter-column security, then any PPT attacker has only a negligible advantage in distinguishing $\text{Game}_{0.2,z}$ from $\text{Game}_{0.2,z-1}$, for each $z$ from 1 to $n-r$.*

*Proof.* We apply the inter-column security game with $k = r+z$, $j = r$, $i^* = r+z$, and $\beta = 1$. To see that the required constraints are satisfied, observe that the $r^{th}$ column always has a 1 in row $i^*$ and slot $\beta$, because this is a below diagonal entry for column $r$. Also observe that column $r$ and column $r+z$ match in all of their entries at or above row $r$, and below row $r$ column $r$ has all 1s. □

We next define $M_3$, which will differ from $M_{2,n-r}$ in that the $r^{th}$ column will now match its specification in $M_L^t$ (note that this only requires a change to the diagonal entry in slot 1). We let $\text{Game}_{0.3}$ denote the variant of the security game with $M_3$ used as the underlying input-activated matrix.

**Lemma B.4.** *If the input-activated obfuscation scheme has inter-column security, then any PPT attacker has only a negligible advantage in distinguishing $\text{Game}_{0.2,n-r}$ from $\text{Game}_{0.3}$.*

*Proof.* Here we apply the inter-column security game with $k = r$, $j = n+1$, $i^* = r$, $\beta = 1$. We note that coming into this transition, column $r$ and the scratch column (column $j$), have all the same values. Additionally, they have a 1 in this slot $\beta$, row $i^*$. Thus, this is a valid invocation of the inter-column game that changes this 1 to 0 in column $r$. As a result, column $r$ will now match its specification in $M_L^t$. □

What now remains is to make the scratch column match $S_t$. For each $z$ from 0 to $n$, we let $M_{3,z}$ denote a matrix that is like $M_3$, except that the first $z$ rows of the scratch column (column $n+1$) match $S_t$. Note that $M_{3,0} = M_3$. We let $\text{Game}_{0.3,z}$ denote a variant of the security game with $M_{3,z}$ used as the underlying input-activated matrix.

**Lemma B.5.** *If the input-activated obfuscation scheme has inter-column security, then any PPT attacker has only a negligible advantage in distinguishing $\text{Game}_{0.3,z}$ from $\text{Game}_{0.3,z-1}$ for each $z$ from 1 to $n$.*

*Proof.* We first note that in any row $z$ where $t_z = 1$ (i.e. the $z^{th}$ bit of $t$ is 1, and this can only occur for $z \leq r$), the scratch column already contained the correct slot values (a 0 in the 0 slot, and a 1 in the 1 slot). So for these values of $z$, the two games are in fact identical, and there is nothing to prove.

It thus suffices to consider values of $z$ such that $t_z = 0$. For such a $z$, note that in $\text{Game}_{3,z-1}$ the scratch column contains 1s in both slots of row $z$. We will apply the inter-column security game to change the value in slot 1 to a 0. This can be done by setting $k = n + 1$, $j = z$, $i^* = z$, and $\beta = 1$. To see that this is a valid invocation, note that the $j^{th}$ column of $M_L^t$ has a 1 for its value in slot 1 on its diagonal entry (row $z$). It has a 0 in its slot 0 at this position, but note that position is exempted from the requirements for the inter-column security game. For rows below the diagonal, row $z$ has all 1 entries, so it certainly dominates the 1 entries in the scratch below this point. For any row $i$ above $z$, if $t_i = 1$ then both column $z$ and the scratch will have 0 in their 0 slots and 1 in their 1 slots for row $i$. If $t_i = 0$, then column $z$ will have both 1s at this row. This means that all the requirements for the inter-column game are satisfied, allowing us to transition from $M_{3,z-1}$ to $M_{3,z}$. □

We now observe that $M_{3,n}$ is equal to the matrix $M_L^t|S_t|M_R^t$ that we were working towards, and hence we have completed this transition for all $t > 0$. To complete the proof of Lemma 4.6, it only remains to deal with the corner case of $t = 0$. For this, we must get from $M_L^{-1}|S|M_R^0$ to $M_L^0|S_0|M_R^0$. We will accomplish this in three steps. We let $M_0 := M_L^{-1}|S|M_R^0$ denote our starting matrix, with $\text{Game}'_0$ denoting the corresponding security game. We define $M_1 := M_L^{-1}|\tilde{S}|M_R^0$, where $\tilde{S}$ is a column of all 1s. We let $\text{Game}'_{0.1}$ denote a variant of the security game where the underlying input-activated matrix is $M_1$. We next define $M_2 := M_L^0|\tilde{S}|M_R^0$. We let $\text{Game}'_{0.2}$ denote a variant of the security game where the underlying input-activated matrix is $M_2$. We let $M_3 := M_L^0|S_0|M_R^0$ denote our target matrix, with $\text{Game}'_{0.3}$ being the corresponding security game.

**Lemma B.6.** *If the input-activated obfuscation scheme has inter-column security, then any PPT attacker has only a negligible advantage in distinguishing between $\text{Game}'_0$ and $\text{Game}'_{0.1}$.*

*Proof.* We invoke the inter-column security game $2n$ times to change each entry of the scratch column from a 0 to 1. Throughout this process, we can set $j = 1$ and $k = n + 1$ and iterate over all of the values of $i^*$ from 1 to $n$ and $\beta = 0, 1$. Each of these invocations is justified by the fact that column 1 in $M_L^{-1}$ contains all 1 entries. □

**Lemma B.7.** *If the input-activated obfuscation scheme has inter-column security, then any PPT attacker has only a negligible advantage in distinguishing between $\text{Game}'_{0.1}$ and $\text{Game}'_{0.2}$.*

*Proof.* Here we invoke the inter-column security game once with $j = n + 1$, $k = 1$, $i^* = 1$ and $\beta = 0$. This invocation is justified by the fact that the scratch column now contains all 1 entries, and so can be used to flip the 0-slot entry of the first column to 0. □

**Lemma B.8.** *If the input-activated obfuscation scheme has inter-column security, then any PPT attacker has only a negligible advantage in distinguishing between $\text{Game}'_{0.2}$ and $\text{Game}'_{0.3}$.*

*Proof.* This transition can be accomplished by $n$ invocations of the inter-column security game. For each $z$ from 1 to $n$, we apply the inter-column security game with $j = z$, $k = n + 1$, $i^* = z$, and $\beta = 1$. We note that column $j = z$ has 1's in all entries except for a 0 in the 0-slot of row $z$. So when we apply the inter-column game with $i^* = z$, this slot is exempted from the requirement

that every 1 in column $k$ must be matched by a 1 in column $j$, thus allowing us to flip the 1-slot of the scratch column in this row from a 1 to 0. Once we have done this for every $z$ from 1 to $n$, the scratch column contains $S_0$. □

This concludes our proof of Lemma 4.6.

# C Proof of Lemma 4.8

For this game transition, the list of programs is fixed, and we are going from an underlying input-activated matrix of $M_L^t|S_t|M_R^t$ to a matrix of $M_L^t|S|M_R^{t+1}$. We will perform this transition gradually over several steps. These steps will only change the values in the $S_t$ and $M_R^t$ portions of the concatenated matrix - the preceding $M_L^t$ portion will just be carried along unchanged throughout.

For notational convenience, we let $M_1 := M_L^t|S_t|M_R^t$. We let $r$ be index of the last 0 in $t$, assuming for now that $t \neq (1, 1, \ldots, 1)$. In other words, $r = n$ if $t_n = 0$, $r = n-1$ if $t_n \neq 0, t_{n-1} = 0$, and so on. For every $z$ from 0 to $n-r$, we define $M_{1,z}$ to be the same as $M_1$, except the *last* $z$ rows of the scratch column have all slots = 1, instead of being $S_t$. For each such $z$, we define $\text{Game}_{2.1,z}$ to be a variant of the security game where the attacker is given an input-activated obfuscation scheme created from $M_{1,z}$ (where we interpret $M_{1,0}$ as $M_1$).

**Lemma C.1.** *For each $z$ from 1 to $n-r$, if the input-activated obfuscation scheme has inter-column security, then any PPT attacker has only a negligible advantage in distinguishing $\text{Game}_{2.1,z}$ from $\text{Game}_{2.1,z-1}$.*

*Proof.* We will invoke the inter-column security game with $k = n+1$, $i^* = n-(z-1)$, $\beta = 1 - t_{n-(z-1)} = 0$, and $j = n+1+n-(z-1)$. To see that this is a valid use of inter-column security, first note that $M_{i^*,j,0} = 1$ for either $M = M_{1,z}$ or $M = M_{1,z-1}$, since this is a diagonal position inside $M_R^t$ and $t_{n-(z-1)} = 1$. Next note that for $i > n-(z-1)$, $M_{i,j,0} = M_{i,j,1} = 1$, since these entries are below the diagonal in $M_R^t$. For all $i < n-(z-1)$, $M_{i,j,t_i} = 1$ holds, by definition of $M_R^t$. Note that these are the only 1 entries in $S_t$ that appear for these rows. Lastly, note that the single slot $M_{n-(z-1),j,1}$ is exempted from the condition that column $j$ have a 1 wherever column $k$ does in the definition of our inter-column security game. □

This brings us to the matrix $M_2 := M_{1,n-r}$. Now for $z$ from 0 to $r-1$, we define $M_{2,r-1-z}$ to be the same as $M_2$, except the first $(r-1)-z$ rows of the scratch column match $S_t$, while the next $z$ rows match column $r$ inside $M_R^t$. For each $z$ from 0 to $r-1$, we define $\text{Game}_{2.2,r-1-z}$ to be a variant of the security game where the attacker is given an input-activated obfuscation scheme created from $M_{2,r-1-z}$. Note that in this part of the proof we are progressing forward by starting with $M_{2,r-1}$ and "counting down" to $M_{2,0}$.

**Lemma C.2.** *For each $z$ from 0 to $r-2$, if the input-activated obfuscation scheme has inter-column security, then any PPT attacker has only a negligible advantage in distinguishing between $\text{Game}_{2.2,r-z-1}$ and $\text{Game}_{2.2,r-z-2}$.*

*Proof.* First we consider what the $r-1-z$ row of $S_t$ and the $r^{th}$ column of $M_R^t$ have in their slots. If $t_{r-1-z} = 0$, then $S_t$ and the $r^{th}$ column of $M_R^t$ both have a 1 value in the 0 slot and a 0 value in the 1 slot on this row. Thus, there is no difference here, and the games are in fact identical.

It thus suffices to consider the case where $t_{r-1-z} = 1$. In this case, $S_t$ has a 0 in the 0 slot and a 1 in the 1 slot, while the $r^{th}$ column of $M_R^t$ has 1's in both slots on this row. So we will invoke the inter-column security game with $k = n + 1$, $i^* = r - 1 - z$, $\beta = 0$, and $j = n + 1 + r - 1 - z$.

To see that this applies, we first observe that $M_{r-1-z,j,0} = 1$, since this is a diagonal position inside $M_R^t$ and $t_{r-1-z} = 1$. We next need to check that for every 1 that currently appears in the $k^{th}$ column, we have a 1 in the same position in the $j^{th}$ column. We first consider the rows $i > r-1-z$. For these rows, the $j^{th}$ column is full of 1's in all slots, as these are below diagonal rows of a column inside $M_R^t$. For rows $i < r - 1 - z$, the $k^{th}$ column currently matches $S_t$, and so only has 1 in the slots corresponding to the $t_i$ values. Note that above diagonal entries of a column inside $M_R^t$ will also have 1's in these slots, so we can indeed apply inter-column security to change the indicated slot in column $k$ from a 0 to a 1. □

We let $\text{Game}_{2.3} := \text{Game}_{2.2,0}$. This brings us to the matrix $M_3 := M_{2,0}$, where the scratch column of $M$ is now exactly what we would like the $r^{th}$ column inside $M_R^t$ to become in order to switch from $M_R^t$ to $M_R^{t+1}$. We let $M_4$ denote the matrix that is formed by adjusting $M_3$ so that the $n + 1 + r$ column matches the $r^{th}$ column in the definition of $M_R^{t+1}$. We define $\text{Game}_{2.4}$ to be a variant of the security game where the attacker is given an input-activated obfuscation scheme created from $M_4$.

**Lemma C.3.** *If the input-activated obfuscation scheme has inter-column security, then any PPT attacker has only a negligible advantage in distinguishing between $\text{Game}_{2.3}$ and $\text{Game}_{2.4}$.*

*Proof.* In fact, the $n + 1 + r$ column inside $M_3$ only differs from this column inside $M_4$ in one position: namely a 0 that appears in the 0 slot in row $r$ that we would like to change to a 1. Clearly, since the scratch column has a 1 in this position, we can use the inter-column security game with $j = n + 1$, $k = n + 1 + r$, $i^* = r$, and $\beta = 0$ to make this transition. □

We now have the $n + 1 + r$ column of $M_4$ matching what it should be in $M_R^{t+1}$. We now must change columns $n + 1 + r + z$ as $z$ ranges from 1 to $n - r$. For each such $z$, we let $M_{4,z}$ denote the matrix which is similar to $M_4$, except that the columns $n + 1 + r$ up to $n + 1 + r + z$ match $M_R^{t+1}$. Note that $M_{4,0} = M_4$. For $z$ from 0 to $n - r$, we define $\text{Game}_{2.4,z}$ to be a variant of the security game where the attacker is given an input-activated obfuscation scheme created from $M_{4,z}$.

**Lemma C.4.** *For each $z$ from 1 to $n-r$, if the input-activated obfuscation scheme has intra-column and inter-column security, then any PPT attacker has only a negligible advantage in distinguishing $\text{Game}_{2.4,z}$ from $\text{Game}_{2.4,z-1}$.*

*Proof.* To transition from $M_{4,z-1}$ to $M_{4,z}$, we consider the $r + z$ column of $M_R^t$. We consider its diagonal slots, i.e. the slots on row $r + z$. We wish to change the value in the 0 slot here from 1 to 0. To do this, we invoke inter-column security with $i^* = r + z$, $\beta = 0$, $k = n + 1 + r + z$, and $j = n + 1 + r$. To see that this applies, note that column $j$ of $M_4$ is equal to the $r^{th}$ column of $M_R^{t+1}$, and this has a 1 in the relevant slot because it is a below diagonal entry. Furthermore, these columns $j$ and $k$ agree in all of their entries above row $i^*$. Now, once we have we this row $i^*$ with both slots having 0 values, we can invoke intra-column security to change the rest of the $n + 1 + r + z$ column to the value it should take in $M_{4,z}$. □

We let $M_5 := M_{4,n-r}$. We note that this is almost the same as the desired $M_6 := M_L^{t+1}|S|M_R^{t+1}$, except that the scratch column has not been reset to all 0 values. For this last transition, we define

$\text{Game}_{2.6}$ as a variant of the security game where the attacker is given an input-activated obfuscation scheme created from $M_6$, and we let $\text{Game}_{2.5} := \text{Game}_{2.4,n-r}$.

**Lemma C.5.** *If the input-activated obfuscation scheme has intra-column and inter-column security, then any PPT attacker has only a negligible advantage in distinguishing $\text{Game}_{2.5}$ from $\text{Game}_{2.6}$.*

*Proof.* In $M_5$, the scratch column is still an exact copy of column $n+1+r$, which is where we left it after using it to perform our switch from $M_R^t$ to $M_R^{t+1}$. Thus, by an easy application of inter-column security with $k = n+1$ and $j = n+1+r$, we can create a row of the final column that has 0's in both slots. Then, finally invoking intra-column security, we can set it back to all 0 values in all rows. □

In the above argument, we assumed that $t < 2^n - 1$. Finally we must argue that we can transition from $M_L^t|S_t|M_R^t$ to $M_L^t|S|M_R^{t+1}$ when $t = 2^n - 1$ (and $t+1 = 2^n$). This will be similar to the argument applied above and will take multiple steps. For notational convenience, we now set $t := 2^n - 1$ and (re-)define $M_1 := M_L^t|S_t|M_R^t$ as our starting point.

For every $z$ from 0 to $n$, we define $M_{1,z}$ to be the same as $M_1$, except the last $z$ rows of the scratch column have all slots = 1, instead of being $S_t$. For each such $z$, we define $\text{Game}'_{2.1,z}$ to be a variant of the security game where the underlying input-activated matrix is $M_{1,z}$.

**Lemma C.6.** *For each $z$ from 1 to $n$, if the input-activated obfuscation scheme has inter-column security, then any PPT attacker has only a negligible advantage in distinguishing $\text{Game}'_{2.1,z}$ from $\text{Game}'_{2.1,z-1}$.*

*Proof.* This follows identically to the proof of Lemma C.1. □

This brings us to the matrix $M_2 := M_{1,n}$, which has a scratch column with all slots = 1. We define $M_3$ to be the same as $M_2$, except that its $n+2$ column also has all slots = 1. We define $\text{Game}'_{2.3}$ to be the variant of the security game where the underlying input-activated matrix is $M_3$.

**Lemma C.7.** *If the input-activated obfuscation scheme has inter-column security, then any PPT attacker has only a negligible advantage in distinguishing $\text{Game}'_{2.1,n}$ from $\text{Game}'_{2.3}$.*

*Proof.* Here we invoke the inter-column security game with $j = n+1$, $k = n+2$, $\beta = 1$, and $i^* = 1$. It is easy to confirm that this applies, as the column $j$ (the scratch column of $M_2$) contains all 1 values. □

We have now transitioned to $M_3$, which is almost equal to the desired $M_4 := M_L^t|S|M_R^{t+1}$, except we want to reset the scratch column of $M_3$ to be all 0 values instead of being all 1 values. We will do this with a final application of the inter-column and then intra-column security games. We let $\text{Game}'_{2.4}$ be a variant of the security game where the underlying input-activated matrix is $M_4$.

**Lemma C.8.** *If the input-activated obfuscation scheme has inter-column and intra-column security, then any PPT attacker has only a negligible advantage in distinguishing $\text{Game}'_{2.3}$ from $\text{Game}'_{2.4}$.*

*Proof.* We invoke the inter-column security game twice with $j = n+2$, $k = n+1$, $i^* = 1$, for $\beta = 0$ and $\beta = 1$. We note that column $j$ contains all 1's, so these invocations allow us to change the first row of the scratch column to have 0's in both slots. Then, by a final application of the intra-column security game, we can change the remaining rows of the scratch column to 0's as well. □

This concludes our proof of Lemma 4.8.

# D Review of Extended CLT Encodings

We begin abstractly, with the generic graded encoding procedures.

## D.1 Graded Encoding Procedures

We recall the definition of a graded encoding system and describe some procedures for graded encodings, mostly following [GGH13a, CLT13, GLW14]. The encodings are similar to [GLW14], with minor differences.

**Definition D.1** (Graded Encoding System)**.** *A $\kappa$-graded encoding system for a ring $R$ is a system of sets $\mathcal{E} = \{\mathcal{E}_i^{(m)} \in \{0,1\}^* : i \in \{0,1,\ldots,\kappa\}, m \in R\}$ with the following properties:*

1. *For every $i$, the sets $\{\mathcal{E}_i^m : m \in R\}$ are disjoint.*
2. *There are binary operations $+$ and $-$ (on $\{0,1\}^*$) such that for every $m_1, m_2 \in R$, every $i$, and every $u_1 \in \mathcal{E}_i^{(m_1)}$ and $u_2 \in \mathcal{E}_i^{(m_2)}$, it holds that $u_1 + u_2 \in \mathcal{E}_i^{(m_1+m_2)}$ and $u_1 - u_2 \in \mathcal{E}_i^{(m_1-m_2)}$ where $m_1 + m_2$ and $m_1 - m_2$ are addition and subtraction in $R$.*
3. *There is a binary operation $\times$ (on $\{0,1\}^*$) such that for every $m_1, m_2 \in R$, every $i_1, i_2$ with $i_1 + i_2 \leq \kappa$, and every $u_1 \in \mathcal{E}_{i_1}^{(m_1)}$ and $u_2 \in \mathcal{E}_{i_2}^{(m_2)}$, it holds that $u_1 \times u_2 \in \mathcal{E}_{i_1+i_2}^{(m_1 \cdot m_2)}$, where $m_1 \cdot m_2$ is multiplication in $R$.*

CLT (and GGH) encodings do not quite meet the definition of graded encoding systems above, since the homomorphisms required in the definition eventually fail when the "noise" in the encodings becomes too large, analogously to how the homomorphisms may eventually fail in lattice-based homomorphic encryption. However, these noise issues are relatively straightforward (if tedious) to deal with.

The set $\mathcal{E}_i^{(m)}$ of encodings of $m$ in $\mathcal{E}_i$ is analogous to the single element $g_i^m \in G_i$ in the algebraic group setting, which encodes $m$ in group $G_i$. We sometimes call encodings in $\mathcal{E}_0$ "level-0 encodings".

Now, we define some procedures for graded encoding schemes.

**Instance Generation** $\mathsf{InstGen}(\lambda, \kappa, \sigma)$ takes as input a security parameter $\lambda$, the multilinearity parameter $\kappa$, a ring dimension parameter $\sigma$, and outputs $(\mathsf{params}, p_{zt})$, where $\mathsf{params}$ is a "description" of a $\kappa$-graded encoding system for a ring $R = R_1 \times \cdots \times R_\sigma$, and $p_{zt}$ is a zero-test parameter for $\mathcal{E}_\kappa$. We assume $R$ is chosen such that the density of zero divisors in each $R_i$ is negligible. We let $\mathsf{esk}$ denote a master secret key associated to the graded encoding system (which is not revealed). A description of $R$ (but not necessarily its decomposition) is included in $\mathsf{params}$.

**Ring Sampler** $\mathsf{Samp}(\mathsf{params})$ is a randomized algorithm that outputs a level-0 encoding of a statistically uniform element $m \in R$, though the encoding itself need not be uniform.

**Re-Randomization** $\mathsf{ReRand}(\mathsf{params}, i, u)$ is a randomized algorithm that takes as input an encoding $u \in \mathcal{E}_i^{(m)}$ for some $m$, and outputs a new encoding $u' \in \mathcal{E}_i^{(m)}$. The distributional requirement is that, for any encodings $u_1, u_2 \in \mathcal{E}_i^{(m)}$, the distributions of $\mathsf{ReRand}(\mathsf{params}, i, u_1)$ and $\mathsf{ReRand}(\mathsf{params}, i, u_2)$ are statistically indistinguishable.

**Remark 1.** *Again, due to the noisiness of GGH and CLT encodings, iterative applications of* $\mathsf{ReRand}$ *will eventually cause the noise to exceed a threshold, after which encodings become garbage. In that context, it will be understood that* $\mathsf{ReRand}$ *must be be correct only "up to noise".*

**Addition, Subtraction, and Multiplication** These are the $+$, $-$, and $\times$ procedures of the graded encoding system.

**Zero-test** The procedure $\mathsf{IsZero}(\mathsf{params}, p_{zt}, u)$ takes an encoding $u$ and outputs "true" if $u \in \mathcal{E}_\kappa^{(0)}$ and "false" otherwise.

Like [GLW14], we define a couple of procedures not provided in [GGH13a, CLT13]. These procedures are directed to graded encoding systems with an encoding space $R$ that can be decomposed nontrivially as a direct product $R_1 \times \cdots \times R_\sigma$.

**Subring Generation** $\mathsf{SubRGen}(\mathsf{esk}, i, S)$ is a private randomized algorithm that takes as input the graded encoding secret key and a subset $S \subset [\sigma]$. It generates $m \in R$ such that $m$ is random in $R_j$ for $j \in S$ but $m$ is 0 in $R_j$ for $j \in [\sigma] \setminus S$. It then outputs a random encoding in $\mathcal{E}_i^{(m)}$.

**Subring Sampling** $\mathsf{SubRSamp}(\mathsf{params}, i, S, \mathcal{G}_{S,i}\})$ takes as input the parameters, $i \in [\kappa]$, $S \subset [\sigma]$, and a "generating set" of encodings $\mathcal{G}_{S,i} \subset \mathcal{E}_{S,i}$, where $\mathcal{E}_{S,i} \subset \mathcal{E}_i$ denotes the subset of encodings that encode some $m$ such that $m$ is 0 in $R_j$ for $j \in [\sigma] \setminus S$.

**Remark 2.** *The form of the "generating set" will depend on the type of encodings. For "noisy" encodings, where multiplying/exponentiating by big numbers would blow up the noise, generating sets instead consist of many random encodings, and we sample by taking a random subset sum of the encodings, and applying the leftover hash lemma to argue that the result is well-distributed.*

## D.2 Overview of CLT Encodings

A $\kappa$-linear symmetric CLT encoding system uses a "small" inner modulus $N = p_1 \cdots p_s$ that is the product of $s = s(\lambda, \kappa)$ "small" primes, and a "large" outer modulus $Q = Q_1 \cdots Q_s$ that is the product of $s$ "large" primes. It uses a random $z \leftarrow \mathbb{Z}_Q^*$. An encoding $c \in \mathcal{E}_1^{(m)}$ is an element of $\mathbb{Z}_Q$ such that

$$c \equiv \frac{[m]_{p_i} + x_i \cdot p_i}{z} \bmod Q_i \text{ for } \ i \in [s], \tag{4}$$

where $[m]_{p_i}$ is $m$ reduced modulo $p_i$ into a small range such as $(-p_i/2, p_i/2)$, and the $x_i$'s are random small integers. An encoding in $\mathcal{E}_T$ has a similar form, but with $z^\kappa$ in the denominator.

For random small integers $h_1, \ldots, h_s$, the system includes a zero-testing parameter $p_{zt}$ for level $\kappa$ of the form:

$$p_{zt} = \sum_{i=1}^{s} h_i \cdot (z^\kappa \cdot p_i^{-1} \bmod Q_i) \cdot \prod_{j \neq i} Q_j \bmod Q. \tag{5}$$

If $c$ is a level-$\kappa$ encoding of $0 \in \mathbb{Z}_N$ – i.e., each $[m]_{p_i} = 0$ – we have:

$$\begin{aligned} c \cdot p_{zt} &= \sum_{i=1}^{s} (x_i \cdot p_i / z^\kappa) \cdot h_i \cdot (z^\kappa \cdot p_i^{-1} \bmod Q_i) \cdot \prod_{j \neq i} Q_j \bmod Q \\ &= \sum_{i=1}^{s} x_i \cdot h_i \cdot \prod_{j \neq i} Q_j \bmod Q \end{aligned}$$

which is a number substantially smaller than $Q$ assuming the $x_i$'s and $h_i$'s satisfy certain smallness constraints – in particular, that each $x_i \cdot h_i \ll Q_i$. On the other hand, if $c$ encodes something other than 0, $c \cdot p_{zt}$ likely will not be a small number, due to uncanceled $p_i^{-1}$'s in the expression above. Thus, $p_{zt}$ enables zero-testing. (Actually, CLT uses a polynomial number of such zero-testing parameters, and they prove that $c$ encodes 0 if it passes the tests with respect to all of them, and does not encode 0 otherwise.)

By CRT, we can add and multiply CLT encodings while preserving their form (per Equation 4) as long as the numerators in Equation 4 do not grow too large – i.e., they do not "wrap" modulo $Q_i$ for any $i$. The $Q_i$'s must be chosen large enough to ensure that such wrapping never occurs for the functions we will compute over the encodings. These additions and multiplications induce additions and multiplications on the underlying "messages" that are encoded, much like homomorphic encryption.

Translating schemes from algebraic multilinear groups to CLT encodings requires some care. In contrast to encodings over groups, CLT encodings are probabilistic and noisy, and come from a distribution. We have to define these distributions, and show that they are correct in our scheme and in the hybrids of our security proof.

Another issue is representing subrings of $\mathbb{Z}_N$ with CLT encodings. There are two big issues here: 1) how to give a useful noise-resilient description of subrings, and 2) whether it is secure in the CLT setting to give descriptions of subrings. The natural way to represent a subring in the no-noise setting is to give a generator of that subring, which typically can be a single element. Then, to generate a random element in that subring, one simply multiplies the generator by a random number. This strategy does not work in the noisy setting, since multiplying an encoding by a big number also blows up the noise. Instead, our approach is to represent a subring by a large "generating set" – a bunch of encodings that encode elements of the subring – and to generate random elements in the subring by taking random subset sums over the generating set and using the leftover hash lemma. Thus, our CLT-based assumptions end up looking somewhat more complicated than the analogous assumptions in the group setting, since each subring generator is expanded into a larger generating set. Regarding security, we have to ask: Is it safe, for example, to give an encoding of some $m$ that is in the index-$p_i$ subring of $\mathbb{Z}_N$? Unfortunately, as mentioned in [GLW14], it is not! As we discuss in more detail in Section D.5, unless one is extremely careful with the parameter settings, one can use such a CLT encoding to efficiently recover $p_i$! The original CLT proposal [CLT13] was careful to never give out encodings in which any divisor of $N$ was "isolated" in this way; for the encodings in the parameters, the encoded values are 0 modulo all of the $p_i$'s or none of them. To translate our composite-order constructions, we need to use subrings, and therefore we cannot use CLT's safe "all-or-nothing" approach. However, we still avoid letting any $p_i$ be "isolated" by giving it many – i.e., $\text{poly}(\lambda)$ – "buddies": any encoding that an attacker sees is 0 modulo $p_i$ and all of its prime buddies $\{p_j\}$, or is (whp) nonzero for all of them. As we discuss in Section D.5, this approach seems resilient to attacks.

Below, we flesh out the overview above. We provide more details on CLT encodings, and on translating our scheme, assumptions and proofs from the general graded encoding setting to CLT.

### D.3 Graded Encoding Procedures for CLT

Here, we describe how the graded encoding procedures from Section D.1 are instantiated in CLT.

**Instance Generation** $\mathsf{InstGen}(\lambda, \kappa, r)$ takes as input the security parameter $\lambda$, the multilinearity parameter $\kappa$, and a ring dimension parameter $\sigma$. It generates, for each $i \in [\sigma]$ and $\theta \in [\Theta = \Theta(\lambda, \kappa)]$, a $\rho = \rho(\lambda, \kappa)$-bit small prime $p_{i,\theta}$ and a $\eta = \eta(\lambda, \kappa)$-bit big prime $Q_{i,\theta}$, and sets $N = \prod_{i,\theta} p_{i,\theta}$ and $Q = \prod_{i,\theta} Q_{i,\theta}$. For $i \in [\sigma]$, we let $R_i$ denote the ring $\mathbb{Z}_{\prod_\theta p_{i,\theta}}$. The parameter $\Theta$ specifies how many primes are associated to each $R_i$, and it needs to be set large (but polynomial) for security reasons. We let $R = R_1 \times \cdots \times R_\sigma = \mathbb{Z}_N$. To eventually obtain correctness and security against known attacks, one can take $\rho = O(\lambda)$, $\eta = O(\lambda \cdot (\lambda + \kappa))$, and $\Theta = (\rho \cdot \eta)^{1+\epsilon}$, $\epsilon > 0$. It generates a value $z \in \mathbb{Z}_Q$.

For parameter $t = t(\lambda, \kappa)$, $\mathsf{InstGen}$ generates $t = t(\lambda, \kappa)$ random numbers $m_j \in \mathbb{Z}_N$, and generates level-0 encodings of them:

$$c_j \equiv [m_j]_{p_{i,\theta}} + x_{ji\theta} \cdot p_{i,\theta} \bmod Q_{i,\theta}.$$

where the $x_{ji\theta}$'s are random numbers in $(-2^\rho, 2^\rho)$. It also generates $t + s$ level-1 encodings of 0:

$$c'_j \equiv \frac{x'_{ji\theta} \cdot p_{i,\theta}}{z} \bmod Q_{i,\theta}.$$

The main requirement on $t$ is that it is large enough to allow application of the leftover hash lemma when we take a subset sum of the $c_j$'s or $c'_j$'s.

It generates a vector $\mathbf{pzt}$ of zero-test parameters, where each $p_{zt}$ in the vector equals $\sum_{i,\theta} h_{i,\theta} \cdot (z^\kappa \cdot p_{i,\theta}^{-1} \bmod Q_{i,\theta}) \cdot \prod_{(i',\theta') \neq (i,\theta)} Q_{i',\theta'} \bmod Q$, where the random $h_{i,\theta}$'s are chosen to have (for example) $3\eta/4$ bits (so that $h_{i,\theta}$ is much smaller than $Q_{i,\theta}$, but $h_{i,\theta}^2$ is much bigger).

It outputs $(\mathsf{params}, \mathbf{pzt})$ where $\mathsf{params}$ includes the basic parameters $\lambda, \kappa, \rho, \eta, s, t, Q, N$. Certain values, such as $z$ and the prime divisors of $N$ and $Q$, remain secret as part of $\mathsf{esk}$.

**Ring Sampler** $\mathsf{Samp}(\mathsf{params})$ generates a random binary vector $b_1 \cdots b_t \in \{0,1\}^t$ and outputs $u \leftarrow \sum_{j=1}^t b_j \cdot c_j \bmod Q$. The statistical uniformity of the encoded value follows by application of the leftover hash lemma to $R = \mathbb{Z}_N$.

**Re-Randomization** $\mathsf{ReRand}(\mathsf{params}, u)$ works by adding a random encoding in $\mathcal{E}_1^{(0)}$ to $u \in \mathcal{E}_1^{(m)}$ to generate a new encoding $u' \in \mathcal{E}_1^{(m)}$. The random encoding of 0 is generated according to a large-enough distribution to "drown" the distribution of $u$. In particular, one sets $u' \leftarrow u + \sum_{j=1}^{t+s} b_j \cdot c'_j \bmod Q$, where $b_1, \ldots, b_t$ are sampled uniformly from $\{0,1\}$ and $b_{t+1}, \ldots, b_{t+s}$ are sampled uniformly from $[-2^\delta, 2^\delta]$ for suitable $\delta = \delta(\lambda, \kappa)$, which can be $\tilde{O}(\lambda)$.

**Remark 3.** *Though we omit details, intuitively the subset sum using the first $t$ encodings of 0 randomizes things "locally" – in particular, it is uniform over the certain cosets of a lattice defined by the last $s$ encodings of 0. The random linear combination over the last $s$ encodings of 0 then randomizes things "globally" by adding a random lattice point drawn uniformly from a huge parallelepiped. CLT [CLT13] proves a "leftover hash lemma over lattices" to establish that this process statically induces the desired canonical distribution.*

**Remark 4.** *In* $\mathsf{ReRand}$*, if $u$ itself was the (perhaps indirect) output of such a randomization procedure, then we would need to make the $b_i$'s even larger to drown the distribution of $u$.*

**Addition, Subtraction, and Multiplication** These operations are performed in the natural way via addition, subtraction, or multiplication modulo $Q$.

**Zero-test** The procedure $\mathsf{IsZero}(\mathsf{params}, \mathbf{pzt}, u)$ takes an encoding $u$ and applies the zero-test parameter to distinguish whether $u \in \mathcal{E}_T^{(0)}$. This procedures works simply by multiplying by each individual $p_{zt}$ and seeing whether the result is small, as described above.

The following procedures are not included in CLT, except for the case of $\sigma = 1$.

**Subring Generation** $\mathsf{SubRGen}(\mathsf{esk}, S)$ is a private randomized algorithm that takes as input a subset $S \subset [\sigma]$ and the graded encoding secret key, which includes $z$ and the factorizations of $N$ and $Q$. It generates $m \in R = \mathbb{Z}_N$ such that $m$ is random in $R_j$ for $j \in S$ but $m$ is 0 in $R_j$ for $j \in [\sigma] \setminus S$. It sets $u_0 \in \mathbb{Z}_Q$ to be $[m]_{p_{i,\theta}}/z \bmod Q_{i,\theta}$. It then sets $u_1 \leftarrow \mathsf{ReRand}(\mathsf{params}, u_0)$.

Observe that when $\sigma = 1$, $\mathsf{SubRGen}(\mathsf{esk}, \{1\})$ outputs an encoding of a random element, while $\mathsf{SubRGen}(\mathsf{esk}, \emptyset)$ outputs an encoding of 0. Both of these functionalities can be performed just using $\mathsf{params}$ (no secret information) in the original version of CLT encodings.

**Subring Sampling** $\mathsf{SubRSamp}(\mathsf{params}, S, \mathcal{G}_S\})$ takes as input the parameters, a subset $S \subset [\sigma]$, and a purported set $\{v_i\}$ of encodings in $\mathcal{E}_S$, where $\mathcal{E}_S$ is the set of encodings that encode that some $m$ such that $m$ is 0 in $R_j$ for $j \in [\sigma] \setminus S$. It runs $\{w_i \leftarrow \mathsf{Samp}(\mathsf{params})\}$, and outputs $u_0 \leftarrow \sum w_i \star v_i$. It then sets $u_1 \leftarrow \mathsf{ReRand}(\mathsf{params}, u_0)$, which should be a statistically random encoding from $\mathcal{E}_S$.

### D.4 Comments on Noise Distributions

The bound on the size of the numerator in CLT encodings grows exponentially with $\kappa$, but this can be accommodated by setting the parameters large enough (but still polynomial).

When adapting the security proofs for our constructions to CLT, we need to ensure that the distributions (in particular, the noise distributions) generated in the security proof are identical to those generated by the encryption algorithm. However, this is easy to ensure. It is clear that the distributions of the encoded terms (in $R$) are statistically indistinguishable. To ensure that the encodings themselves are statistically indistinguishable, we can modify the encryption procedure to apply the same $\mathsf{ReRand}$ algorithm that is used in the proof, with parameters sufficient to "drown out" the distribution of the initial encoding output by encryption.

### D.5 Discussion of the Assumptions

As mentioned above, the most "natural" way to translate a scheme over groups of composite order $N = p_1 \cdots p_\sigma$ to CLT encodings would be to use $\mathbb{Z}_N$ directly as the CLT encoding space. Unfortunately, this approach fails for security reasons (or at least the security seems much more tenuous than for conventional CLT encodings).

Suppose we use the approach above, and we obtain an encoding $c$ in $\mathcal{E}_\kappa$ of some $m \in \mathbb{Z}_N$ such that $[m]_{p_j} \neq 0$ but $[m]_{p_i} = 0$ for all $i \neq j$. (We stress that the original CLT proposal [CLT13] does not give out encodings of this form, and thus is not subject to this attack.) That is,

$$c = \frac{[m]_{p_j} + x_j \cdot p_j}{z^\kappa} \bmod Q_j, \qquad c = \frac{x_i \cdot p_i}{z^\kappa} \bmod Q_i \text{ for } i \neq j$$

for fairly small $x_j$ and $\{x_i\}$. Set $a \leftarrow c \cdot p_{zt} \bmod Q$. We have that

$$a = ([m]_{p_j} \cdot h_j \cdot p_j^{-1} \mod Q_j) \cdot \prod_{k \neq j} Q_k + \sum_{i \in [s]} x_i \cdot h_i \cdot \prod_{k \neq i} Q_k \bmod Q.$$

We do not have $p_j$ a priori, but we do know that

$$b := a \cdot p_j = [m]_{p_j} \cdot h_j \cdot \prod_{k \neq j} Q_k + \sum_{i \in [s]} p_j \cdot x_i \cdot h_i \cdot \prod_{k \neq i} Q_k \bmod Q.$$

Furthermore, if the parameters are such that the values $[m]_{p_j} \cdot h_j$ and $\{p_j \cdot x_i \cdot h_i\}$ are all small in relation to the $Q_i$'s, then the value $b$ is small in relation to $Q$. (Recall that the values $\{x_i \cdot h_i\}$ should be small to allow correct zero testing.) If this value is small enough, we can recover $p_j$ via lattice reduction.

Specifically, let $B$ be an approximation of the value $b/p_j$, based on the distributions of variables. (Note that $B$ is dominated by the summation $\sum_{i \in [s]} x_i \cdot h_i \cdot \prod_{k \neq i} Q_k \bmod Q$, and thus is comparable in size to the small mod-$Q$ value that one would obtain by zero-testing an encoding that actually encodes 0.) Consider the two-dimensional lattice $L$ generated by the vectors $(B, a)$ and $(0, Q)$. This lattice contains the vector $\vec{v} := (B \cdot p_j, b)$, of length approximately $B \cdot p_j \cdot \sqrt{2}$. (Let us assume its length is upper-bounded by $2Bp_j$.) On the other hand, the determinant of the lattice is $B \cdot Q$, implying that all vectors in $L$ that are not parallel to $\vec{v}$ must have length at least $B \cdot Q/(2Bp_j) = Q/2p_j$. If $B < Q/4p_j^2$, then $2Bp_j < Q/2p_j$, and $\vec{v}$ is the unique shortest vector in $L$, which can be recovered easily via lattice reduction. Recovery of $\vec{v}$ means recovery of $p_j$, a devastating attack on the encodings. While in principle the parameters could conceivably be set carefully to ensure that $B$ is comfortably larger than $Q/4p_j^2$ (to avoid this attack) and comfortably smaller than $Q$ (to allow zero-testing), we note that nothing approaching this amount of care was needed when setting the parameters of the original CLT scheme.

The attack can be extended, to some extent, to encodings of $m \in \mathbb{Z}_N$ such that $m$ is nonzero modulo more than one prime divisor of $N$. For example, suppose that we have an encoding of $m$ where $m$ has nonzero residues modulo just $p_j$ and $p_k$. Then, one can define $a$ as above, $b$ as $[a \cdot p_j \cdot p_k]_Q$, $B$ as an approximation of $b/p_j p_k$, and reduce the lattice formed by $(B, a)$ and $(0, Q)$. However, the target vector $(B \cdot p_j \cdot p_k, b)$ in this lattice will not be as short (it will be about $p_k$ times longer), and thus $B$ needs to be correspondingly smaller for lattice reduction to be effective. In general, it seems, the greater the nonzero support of $m$, the harder it is to use an encoding of $m$ to recover factors of $N$.

Indeed, this is consistent with the security concept for CLT encodings. Consider a CLT encoding scheme like [CLT13], but where $N$ and $Q$ are each divisible by only two primes. Such an encoding scheme would be subject to essentially the same attack as above. CLT eliminates such attacks by using many primes, only revealing encodings that are zero with respect to all of the primes or none of them, and gluing together zero-testers for individual primes into an aggregate zero tester. To extend CLT encodings so that we can reveal encodings of elements that are in subrings, we use a similar concept: we associate many primes to each subring $R_i$, and apply CLT's "all-or-nothing" approach within each subring: encodings are zero with respect to all of the primes associated to $R_i$ or none of them.