New Multilinear Maps from CLT13 with Provable Security Against Zeroizing Attacks

Fermi Ma* and Mark Zhandry**

Princeton University

Abstract. We devise the first weak multilinear map model for CLT13 multilinear maps (Coron et al., CRYPTO 2013) that captures *all known* classical polynomial-time attacks on the maps. We then show important applications of our model. First, we show that in our model, several existing obfuscation and order-revealing encryption schemes, when instantiated with CLT13 maps, are secure against known attacks under a mild algebraic complexity assumption used in prior work. These are schemes that are actually being implemented for experimentation. However, until our work, they had no rigorous justification for security.

Next, we turn to building *constant degree* multilinear maps on top of CLT13 *for which there are no known attacks*. Precisely, we prove that our scheme achieves the ideal security notion for multilinear maps in our weak CLT13 model, under a much stronger variant of the algebraic complexity assumption used above. Our multilinear maps do not achieve the full functionality of multilinear maps as envisioned by Boneh and Silverberg (Contemporary Mathematics, 2003), but do allow for re-randomization and for encoding arbitrary plaintext elements.

1 Introduction

Cryptographic multilinear maps have proven to be a revolutionary tool. Very roughly, a multilinear map is an encoding scheme where one can blindly compute polynomials over encoded elements, without any knowledge of the underlying elements. They have been used for numerous cutting-edge cryptographic applications, such as multiparty non-interactive key agreement [BS03], attribute-based encryption for circuits [GGH$^+$13c], asymptotically optimal broadcast encryption [BWZ14a], witness encryption [GGSW13], functional encryption [GGH$^+$13b, GGHZ16], and most notably mathematical program obfuscation [GGH$^+$13b]. In turn, obfuscation has been used to construct even more amazing applications [SW14, HSW14, BZ14, PPS15, CLP15, HW15, AS16, BPW16] as well as establish interesting connections to other areas of computer science [BZ16, BPR15].

Unfortunately, all known multilinear maps for degree $d > 2$ [GGH13a, CLT13, GGH15] have suffered from devastating attacks known as "zeroizing" attacks [GGH13a, CHL$^+$15, HJ16, CLLT16a]. These attacks have rendered most of the applications above insecure. In response, many authors introduced "fixes" for the multilinear maps. These fixes came in many forms, from tweaking how information is extracted from the map [CLT15] to compiling the existing weak multilinear maps into new ones that were presumably stronger [GGHZ16, BWZ14b, Hal15]. However, these fixes were largely ad hoc, and indeed it was quickly shown how to generalize the attacks to the fixes [BWZ14b, CGH$^+$15, BGH$^+$15, Hal15, CFL$^+$16].

Given these many attacks and the speed at which fixes were subsequently broken, researchers have begun attempting to build applications in a sound way using weak maps. The initial observation by Badrinarayanan et al. [BMSZ16] is that all (classical polynomial-time[1]) attacks require the ability to obtain an encoding of zero. Miles, Sahai, and Zhandry [MSZ16] observed moreover that all attacks on the original GGH13 multilinear map have a very similar structure. They define an abstract attack model, called "annihilating

* fermima1@gmail.com

** mzhandry@princeton.edu

[1] Sub-exponential and quantum attacks have been discovered on the multilinear maps [CDPR16, ABD16, CJL16]. In this work we will focus on classical adversaries, and will not consider quantum attacks. We will also not consider sub-exponential attacks as a break, since they can be defeated by increasing the security parameter

attacks", that encompasses and generalize all existing attacks on these specific maps. Remarkably, since their initial publication, all subsequent attacks found on GGH13 have fit within the annihilating model. Therefore, this annihilating model appears to be a fully general abstraction of known attack techniques for GGH13 multilinear maps.

Badrinarayanan et al. [BMSZ16] and Garg et al. [GMM+16] then show how to construct witness encryption, obfuscation, and order-revealing encryption that is provably secure in this weak multilinear map model, and hence secure against all known attacks on GGH13. To date, these are the only *direct* applications of multilinear maps that have been proved secure in the weak multilinear map model for GGH13[2]. Moreover, GGH13 is the only multilinear map for which an accurate weak model has been devised. This leads to the following goals:

Devise weak multilinear map models for other multilinear maps — such as CLT13 or GGH15 — that capture all known attack strategies on the maps

Show more direct applications of multilinear maps that can be constructed and proved secure in weak multilinear map models

A weak multilinear map model for CLT13 is especially important, as it is currently the most efficient multilinear map known [LMA+16], and therefore most likely to eventually become usable in practice.

In addition to developing techniques for using existing weak multilinear maps for applications, researchers are also trying to solve the following:

Construct multilinear maps that are not vulnerable to zeroizing attacks

There has been some initial progress toward this goal. Several authors [PS15, AFH+16] have shown how to construct some version of multilinear maps from obfuscation, which can in turn be built from weak multilinear maps using the constructions above. This gives a compelling proof of concept that multilinear maps immune to zeroizing attacks should be possible. However, as obfuscation is currently incredibly inefficient, such multilinear map constructions are entirely impractical today. Moreover, obfuscation can be used to directly achieve most applications of multilinear maps, so adding a layer of multilinear maps between obfuscation and application will likely compound the efficiency limitations. Therefore, it is important to build multilinear maps without obfuscation.

1.1 Our work: New Multilinear Maps

In this paper, we make additional progress on all three goals above. We revisit the idea of "fixing" multilinear maps through using weak multilinear maps to build strong maps. Unlike the obfuscation-based constructions above, we do not use obfuscation, though our construction is based on obfuscation techniques. Unlike the fixes discussed above that were quickly broken, we develop our fix in a methodical way that allows us to formally argue our fix is immune to generalizations of zeroizing attacks. Specifically, our results are the following:

- First, we need a framework in which to argue security against zeroizing attacks. Our first result is a new weak multilinear map model for CLT13 maps. We demonstrate that this model naturally captures all known attack strategies on CLT13. The model is somewhat different than the model for GGH13 maps, owing to the somewhat different technical details of the attacks.
- To aid the analysis of schemes in our model, we prove that an attack in the weak CLT13 model requires the existence of a certain type of "annihilating polynomial," analogous to the annihilating polynomials in the weak GGH13 model, but simpler. As an immediate consequence, we can prove the security of an *existing* class of obfuscation constructions [BGK+14, BMSZ16] in the weak CLT13 model, under the same algebraic complexity assumption as in [GMM+16]. As these obfuscation constructions are currently being implemented [LMA+16], it is important to justify the security of these constructions over CLT13. Note that [BGK+14, BMSZ16] are *not* secure in the weak GGH13 model, indicating that the weak CLT13 model may be somewhat more useful.

[2] Most other applications are possible by using obfuscation as a building block

– Armed with a weak model for CLT13, we devise a new *constant-degree* multilinear map scheme built on top of CLT13. We then prove that within our weak CLT13 model, *there are no attacks on our new scheme*. That is, any attack at all on our scheme will yield an attack that does not fit in our CLT13 model, and hence gives a brand-new attack technique on CLT13 maps. Our scheme is based on obfuscation techniques, but avoids building a full obfuscation scheme, making our scheme more efficient than obfuscation-based multilinear maps, at least for simple settings.
  Concretely, for a trilinear map with 128 bits of security, one of our elements will consist of about 2×10^5 CLT13 group elements using a degree-57 CLT13 map. Addition and multiplication will involve about 2×10^5 and 3×10^7 CLT13 operations, respectively. For a more aggressive 80 bits of security, one of our elements will consist of about 7×10^4 group elements using a degree-37 CLT13 map. Addition and multiplication will involve about 7×10^4 and 5×10^6 CLT13 operations, respectively. We did not calculate the size public parameters of our scheme, but they will be a couple orders of magnitude larger than each of our encodings. Our encoding size will likely be larger than what is possible with obfuscation, and our scheme is certainly not practical. However, the running times for group operations will be far, far better than what is possible with current obfuscation constructions.
  The lack of a zeroizing attack means we can be more liberal in the types of encodings that are made public. This allows for greatly enhanced functionality compared to existing multilinear map schemes: for example, we can encode arbitrary ring elements and give out encodings of zero.
  There are some notable limitations of our construction and analysis, however. First, for maximum functionality and for our analysis to apply, we must restrict the degree of our multilinear maps to be constant. Note that if one is willing to sacrifice some functionality and have a more heuristic argument for security, then our maps naturally extend to arbitrary polynomial degree. The other main limitation is that our security proof relies on a new algebraic complexity assumption about annihilating polynomials. The assumption is similar to the one used in [GMM+16], though our new assumption is somewhat stronger and less justified than theirs.
  While our scheme is far from practical, we believe this result is a proof of concept that multilinear maps without zeroizing attacks are possible without first building obfuscation. Hopefully, future work will be able to streamline our construction to obtain much more efficient multilinear maps.
– Despite some functionality limitations, our multilinear maps can still be used to solve problems that were not previously possible without first building obfuscation. For example, we show how to use it for multiparty non-interactive key exchange (NIKE) for a constant number of users. This is the most efficient scheme for $n > 3$ users that is immune to known attacks. Hopefully, our maps can be user to make other applications much more efficient as well.
– We note that our techniques for constructing multilinear maps from CLT13 can be combined with the techniques of Garg et al. [GMM+16] to give new multilinear maps in the weak model for GGH13.

1.2 Techniques

Weak CLT13 Model In CLT13, there is a composite modulus $N = \prod_i p_i$ [3]. An encoding s is an integer mod N. Let $s_i = s \bmod p_i$ be the vector of Chinese Remaindered components. Each component s_i encodes a component m_i of the plaintext element. The plaintext space can therefore be interpreted as a vector of integers. Each encoding is associated to a level, which is a subset of $\{1, \ldots, d\}$, where d is the multilinearity of the map. Encodings can be added and multiplied, following certain level-restrictions, until a "top-level" encoding is obtained, which is an encoding relative to the set $\{1, \ldots, d\}$. For singleton sets, we will drop the set notation, and let level $\{i\}$ be denoted as level i.

In CLT13, if s is a top-level encoding of zero — meaning all components of the plaintext are 0 — then one can obtain from it $t = \sum_i \gamma_i s_i$, where γ_i is a rational number, and equality holds over the rationals. γ_i are unknown, but global constants determined by the parameters of the scheme. That is, for each s, the derived t will use the same γ_i.

[3] In [CLT13], this modulus is referred to as x_0

Attacks on CLT13 multilinear maps all follow a particular form. First, public encodings are combined to give top-level encodings, *using operations explicitly allowed by the maps.* If these top-level encodings are zero, then one obtains a t term. The next step in the attack is to solve a polynomial equation Q where the coefficients are obtained from the t terms. In current attacks, the polynomial equation is the characteristic polynomial of a matrix whose entries are rational functions of the zeros.

The next step is to show that the solutions to Q isolate the various s_i components of the initial encodings. Then by performing some GCD computations, one is then able to extract the prime factors p_i, which leads to a complete break of the CLT13 scheme. We show how to capture this attack strategy, and in fact much more general potential strategies, in a new abstract attack model for CLT13. Our model is defined as follows:

- Denote the set of encodings provided to the adversary as $\langle s \rangle$, where each s encodes some plaintext element.
- The adversary is allowed to combine the encodings as explicitly allowed by the multilinear map. Operations are performed component-wise on the underlying plaintext elements.
- If the adversary ever gets a top-level encoding of zero — meaning that the plaintext element is zero in all coordinates — this zero is some polynomial p in the underlying plaintext elements. The adversary obtains a handle to the corresponding element $t = \sum_i \gamma_i p(\langle s \rangle_i)$. Here, $\langle s \rangle_i$ represents the collection of ith Chinese Remaindering components of the various encodings provided.
- The adversary then tries to construct a polynomial Q_i that isolates the ith component for some i. The way we model a successful isolation is that Q_i is a polynomial in two sets of variables: T variables — which correspond to the t terms obtained above — and S variables — which correspond to the ith components of the s encodings. The adversary's goal is to devise a polynomial Q_i such that Q evaluates to 0 when S is substituted for $\langle s \rangle_i$ and T is substituted for the set of t obtained above. We say the adversary wins if it finds such a Q.

In a real attack, roughly, the adversary takes Q, plugs in the values of t, and then solves over the rationals for $\langle s \rangle_i$. Then by taking $GCD(N, s - s_i)$ for some encoding s, she obtains the prime factor p_i. The real adversary does this for every i until it completely factors N. In general, solving Q for $\langle s \rangle_i$ is a computationally intractable task and may not yield unique solutions. The attacks in the literature build a specific Q that allows them to solve efficiently. In our model, we conservatively model *any* Q the adversary can find, even ones that are intractable to solve, as a successful attack.

We note that the attacks described in the literature actually build a Q that is a rational function. However, such rational functions can readily be converted into polynomial functions. We indeed demonstrate that such a polynomial Q is implicit in all known attack strategies.

Next, we prove that an attack in our CLT13 model actually yields an attack in a much simpler model that we call the "plain annihilating model." Here, the adversary still constructs polynomials p of the underlying encodings, trying to find a top-level zero. However now, instead of trying to find a Q_i as above, the adversary simply tries to find a polynomial R that annihilates the p polynomials. That is, $R(\{p(\langle S \rangle)\}_p)$ is identically zero, where $\langle S \rangle$ are now treated as formal variables.

With this simpler model in hand and using analysis from prior work [BMSZ16], we immediately obtain the VBB-security of existing obfuscation constructions [BGK+14, BMSZ16] based on branching programs. Those works show that the only top-level zeros that can be obtained correspond to the evaluations of branching programs. Therefore, relying on the Branching Program Unannihilatability Assumption (BPUA) of [GMM+16], we find that it is impossible to find an annihilating polynomial R, and hence a polynomial Q_i. This gives security in our CLT13 model.

New Multilinear Maps We now turn to developing a new multilinear map scheme that we can prove secure in our weak model for CLT13. In this work, we focus on building an asymmetric scheme, where levels are subsets of $\{1, \ldots, d\}$, and elements can only be multiplied if they belong to disjoint levels. At this time we do not know how to extend our results to construct symmetric multilinear maps, though we believe our techniques will be useful in that setting. The scheme will be based heavily on obfuscation techniques, plus

some new techniques that we develop, but will not build a full obfuscation scheme. Therefore, we expect that our multilinear maps will be much more efficient than those that can be built using obfuscation.

Our starting point is Garg et al. [GGHZ16][4], which offered a potential fix to block zeroizing attacks, which we will call the GGH16 fix. The fix was quickly broken, but we show how to further develop the idea into a complete fix. Garg et al. define a level-i "meta-encoding" of x to be a matrix of level-i CLT13 encodings, obtained by encoding component-wise matrices of the form:

$$R \cdot \begin{pmatrix} x & 0 & \dots & 0 \\ 0 & \$ & \cdots & \$ \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \$ & \cdots & \$ \end{pmatrix} \cdot R^{-1}$$

where \$ represent plaintexts drawn independently at random[5], and R is a random matrix of plaintext elements.

Such meta-encodings can be added and multiplied just like CLT13 encodings, since the matrices R cancel out. However, due to the R matrices, it is no longer possible to isolate the upper-left corner to perform a zero-test on x. Instead, also handed out are "bookend" vectors s, t which encode the plaintext vectors

$$\begin{pmatrix} 1 & 0 & \cdots & 0 \end{pmatrix} \cdot R^{-1} \text{ and } R \cdot \begin{pmatrix} 1 & 0 & \cdots & 0 \end{pmatrix}^T \ ,$$

respectively. Now by multiplying a meta-encoding by the bookend vectors on the left and right, one obtains a CLT13 encoding of the plaintext x, which can then be zero-tested.

Next, Garg et al. include with the public parameters meta-encodings of various powers of 2, as well as many meta-encodings of 0. Powers of 2 allow for anyone to encode arbitrary elements, and the encodings of 0 allow for re-randomizing encodings. Unfortunately, as shown in [CGH$^+$15], this fix does not actually protect against zeroizing attacks: with a bit more work, the meta-encodings of 0 can be used just like regular encodings of zero in the attacks to break the scheme.

To help motivate our new scheme, think of the GGH16 fix as follows: arrange the matrices in a grid: the columns of the grid correspond to the levels, and the matrices for level i are listed out in column i in an arbitrary order. We will call a "monomial" the product of one meta-encoding from each level, in level order (e.g. the level-1 encoding comes first, then level-2, etc). Such monomials correspond to an iterated matrix product that selects one matrix from each column. We re-interpret these monomials as evaluations of a certain branching program. In this branching program, there are t inputs, and each input is not a bit, but a digit from 0 to $k-1$ where k is the number of matrices in each column. Each input digit selects the matrix from the corresponding column, and the result of the computation is the result of the corresponding iterated matrix product.

Note that this branching program is *read-once*, and this is fundamentally why the fix does not succeed. Intuitively, this is because a read-once branching program can be annihilated, in the sense that it is possible to construct a set of inputs and an annihilating polynomial Q such that Q always evaluates to zero on the set of branching program outputs. For example, one can partition the input bits into two sets, and select subsets S at T of partial inputs from each half of the input partition. Evaluate the branching program on all points in the combinatorial rectangle defined by S, T, and arrange as a matrix. The rank of this matrix is at most the width of the branching program. Therefore, as long as the number of partial inputs is larger than the width, this branching program will be annihilated by the determinant. This is true for arbitrary branching programs, not just the branching programs derived above.

One possible way to block the attack above is to make the branching program so wide that even if the adversary queries on the entire domain, the matrix obtained above is still full rank. While it is possible to

[4] We actually need the version of [GGHZ16] dated November 12, 2014 from https://eprint.iacr.org/eprint-bin/versions.pl?entry=2014/666. More recent versions and the proceedings version removed the CLT13 fix that we start from

[5] Actually, in [GGHZ16], some of the zeros are also set to be random elements, but the above form suffices for our discussion and more naturally leads to our construction

do this to build a constant degree multilinear map over CLT13, the map will be of little use. Roughly, the reason is that the branching program is now so wide that adding a random subset-sum of zero encodings is insufficient to fully re-randomize.

Instead, we turn to Garg et al. [GMM+16]'s obfuscator, which blocks this annihilating attack for obfuscation by explicitly requiring the branching program being obfuscated to read each input many times. By reading each input multiple times, the rank of the matrix above grows exponentially, blocking determinant-style attacks. Moreover, under the assumption that there are PRFs that can be computed by branching programs, such read-many programs cannot be annihilated in general. Garg et al. therefore conjecture a branching program un-annihilatability assumption, which says that read-many branching programs cannot be annihilated. Under this assumption, Garg et al. prove security in the weak GGH13 model.

Inspired by this interpretation and by techniques used to prove security of obfuscation, we modify the GGH16 fix to correspond to a read-many branching program. This will allow us to block determinant-style attacks without increasing the width, allowing for re-randomization. Toward that end, we associate each meta-level i with ℓ different CLT13 levels, interleaving the levels for different i. This means that for an d-level meta-multilinear map, we will need $d\ell + 2$ CLT13 levels (the extra 2 levels for the bookends). A meta-encoding at level i will be a sequence of ℓ different matrices of encodings, where the ℓ matrices are encoded at the ℓ corresponding CLT13 levels. The matrices have the form:

$$R_{i-1} \cdot \begin{pmatrix} x & 0 & \dots & 0 \\ 0 & \$ & \cdots & \$ \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \$ & \cdots & \$ \end{pmatrix} \cdot R_i^{-1}, \quad R_{d+i-1} \cdot \begin{pmatrix} 0 & 0 & \dots & 0 \\ 0 & \$ & \cdots & \$ \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \$ & \cdots & \$ \end{pmatrix} \cdot R_{d+i}^{-1}, \quad \cdots \quad , \quad R_{(\ell-1)d+i-i} \cdot \begin{pmatrix} 0 & 0 & \dots & 0 \\ 0 & \$ & \cdots & \$ \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \$ & \cdots & \$ \end{pmatrix} \cdot R_{(\ell-1)d+i}^{-1}$$

Essentially, each of our meta-encodings is a list of GGH16 meta-encodings, where the first meta-encoding encodes x, and the rest encode 0. Our bookend vectors have the form $\begin{pmatrix} 1 & \$ & \cdots & \$ \end{pmatrix} \cdot R_0^{-1}$ and $R_{d\ell} \cdot \begin{pmatrix} 1 & \$ & \cdots & \$ \end{pmatrix}^T$ and are encoded, respectively, in the two remaining CLT13 levels. Unlike GGHZ16, we will not have the \$ terms be random, but instead chosen more carefully (details below). Note that we chose different randomizing matrices R in each position; this corresponds to the randomizing matrices used for Kilian [Kil88] randomization of branching programs in obfuscation. Such randomization forces matrices to be multiplied in order as in a branching program.

Addition is component-wise. For this discussion, we will only allow a pairing operation that goes directly to the top level; this is the kind of multilinear map envisioned by [BS03]. We explain how to give intermediate levels below.

The pairing operation takes one meta-encoding for each meta-level, and arranges the all the matrices in branching-program order. Then, roughly, it multiplies the matrices together, along with the bookends. The result is a single top-level CLT13 encoding, which can be zero-tested as in CLT13. We have to slightly tweak the procedure scheme for this to work, as multiplying all the matrices of a top-level encoding will always give an encoding of zero (since the upper left corner of most of the matrices is 0). Instead, we add an offset vector mid way through the pairing operation to make the CLT13 encoding an encoding of the correct value; see Section 5 for details. For the purposes of this discussion, however, this tweak can be ignored.

The good news is that if one restricts to adding and pairing encodings as described, this blocks the zeroizing attack on GGH16 meta-encodings, assuming ℓ is large enough. We would now like to prove our scheme is actually secure, using the branching program un-annihilatability assumption as was done in obfuscation. Unfortunately, there as several difficulties here:

– First, we need to force the adversary to follow the prescribed pairing procedure. While the pairing operation is basically just a branching program evaluation, this ends up being quite different than in the setting of obfuscation. For example, in obfuscation, forcing input consistency can be done with the level structure of the underlying multilinear map. In our case, this appears impossible. The reason is that we want to be able to add two encodings at the same meta-level before pairing, meaning the underlying encodings must be at the same CLT13 level. In obfuscation, the different encodings for a particular input

are encoded at different levels. There are other ways to force input consistency [GGH+13b, BR14], but they appear to run into similar problems.
- Second, the ability to add encodings means we cannot quite interpret allowed operations as just evaluations of a branching program. For example, if one adds two meta-encodings, and then multiplies them by a third, the result is a linear combination of iterated matrix products containing *cross terms* of the branching program that mix inputs. This is a result of the degree of zero-testing being non-linear. Certain cross terms do not correspond to any branching program evaluation. Therefore, prior means of forcing input consistency will be too restrictive for our needs.

We overcome these issues by developing several new techniques:

- First, we prove a generalization of a lemma by Badrinarayanan et al. [BMSZ16] which tightly characterize the types of iterated matrix products that an adversary is allowed to create. Our lemma works in far more general settings so as to be applicable to our scheme.
- Second, we re-interpret the allowed operations not as branching program evaluations, but as *vector-input* branching program evaluations, a new notion we define. In a vector-input branching program, inputs are no longer digits, but a list of vectors. The vectors specify a linear combination. To evaluate, for each column apply the corresponding linear combination to the matrices in that column, and then multiply all the results together. By being able to take linear combinations of the input matrices, we now capture the ability of an adversary to add encodings.
- Finally, we introduce new "enforcing" matrices that we place in the \$ entries. The goal of our enforcing matrices is to force the adversary's operations to correspond to vector-input branching program evaluations.

Using our enforcing matrices and our new analysis techniques, we show that the adversary is limited to producing linear combinations of vector-input branching program evaluations. Therefore, if the adversary can attack in our weak CLT13 model, it can find an annihilating polynomial for vector-input branching programs. We therefore conjecture that, like regular branching programs, vector-input branching programs cannot be annihilated. Under this assumption, no zeroizing attacks exist on our scheme.

Discussion. We now discuss some limitations of our construction above.

- We do not know how to justify our vector-input branching program assumption based on PRFs, unlike the corresponding assumption for standard branching programs. The reasons are twofold:
  - Most importantly, we do not know of any PRFs that can be evaluated by vector-input branching programs.
  - In our analysis, the adversary may produce a linear combination of *exponentially many* vector-input branching program evaluations. Therefore, an annihilating polynomial annihilates exponentially-many inputs, and therefore would not correspond to a polynomial-time attack on a PRF, even if one were computable by vector-input branching programs. It may be possible that this second limitation can be overcome using a sub-exponentially secure PRF, provided that the PRF is computable by a vector-input branching program. Furthermore, we hope that this second limitation is a limitation of our analysis and not a fundamental problem, giving hope that subsequent work can remove this limitation.

  Nonetheless, the only annihilating polynomials we could find for vector-input branching programs are determinant polynomials as described above, which are also the only annihilating polynomials we know of for plain branching programs. Therefore, it seems reasonable at this time to conjecture that determinants are the only annihilating polynomials. If this conjecture holds, then any annihilating polynomial will require circuits of size roughly w^ℓ, where w is the width of matrices in the branching program. By setting 2^ℓ to be 2^λ for desired security parameter λ, this will block known attacks.
- Our discussion above showed immunity to zeroizing attacks. However, we still need to show immunity to direct attacks that just work by combining components as allowed in the plain generic multilinear map model. Basically, we need to be able to simulate any query in the generic multilinear map model for

CLT13, by making only generic multilinear map queries for our meta-scheme. This is unfortunately problematic, due again to the fact that our analysis allows the adversary to come up with exponential-sized linear combinations. We are able to show, in the case of constant-degree multilinear maps, that such a simulator can be built. Intuitively, this is because any constant-degree polynomial has only polynomially-many monomials, and can therefore be represented efficiently. While our simulator is query efficient, it unfortunately is computationally inefficient. This means that any derived adversary for CLT13 will be query efficient in our CLT13 model, but computationally inefficient. This still allows us to justify most assumptions, such as multilinear DDH. However, it may present an issue if our scheme will be combined with other cryptographic primitives.

- Our discussion above only allows for directly pairing to the top level. However, it is easy to define pairing operations for intermediate levels. Adjacent levels (say 1,2) can easily be paired by simply constructing ℓ matrices that are the pairwise products of the ℓ matrices in the two levels. Due to using different Kilian randomization between each matrix, we cannot directly pair non-adjacent levels, say 1 and 3. For non-adjacent levels, instead of matrix multiplications, we *tensor* the encodings, generating all degree 2 monomials. This tensoring can also be extended to higher levels. This greatly expands the size of encodings; for constant-degree encodings, however, the size of encodings remains polynomial.
- The main limitation of our new multilinear map construction is that it is not possible to multiply an encoding by scalar. One could try repeated doubling, but our scheme inherits the noisiness of CLT13, and this repeated doubling will cause the noise to increase too much. One potential solution is that an encoding of x actually consists of encodings of $x, 2x, 4x, 8x$, etc. Now instead of repeated doubling, multiplying by a scalar is just a subset sum. Of course, this operation "eats up" the powers of 2, so it can only be done a few times before one runs out of encodings.
  Another potential option, depending on application, is to introduce additional "dummy" levels. To multiply by a scalar, first encode it in the "dummy" level, and then pair with the element. This of course changes the level at which the element is encoded, but for some applications this is sufficient. Below, we show how to use this idea to give a multiparty NIKE protocol for a constant number of users.

Multiparty Non-Interactive Key Exchange (NIKE) Here, we very briefly describe how to use our new multilinear maps to construct multiparty NIKE. The basic scheme shown by Boneh and Silverberg [BS03] will not work because (1) they need a symmetric multilinear map, and (2) they need to be able to multiply encodings by ring elements. We show how to tweak the scheme to work with an asymmetric map that does not allow multiplying encodings by ring elements. For d users, instantiate our scheme with d levels, one more than is needed by [BS03].

User i chooses a random ring element a_i, and then computes encodings $[a_i]_u$ of a_i at every singleton level u. User i publishes all the encodings (after re-randomizaiton), *except* the encoding at level 1.

Upon receiving the encodings from all other users, user i arbitrarily assigns each of the other $d-1$ users to the levels $2, \ldots, d$. Let u_j be the level assigned to user j. Then it pairs its private elements $[a_i]_1$ together with $[a_j]_{u_j}$ for each $j \neq i$. The result is an encoding of $\prod_j a_j$ at the top level. Everyone computes the same encoding, which can be extracted to get the shared secret key.

Meanwhile, an adversary, who never sees an encoding of a_i at level 1, cannot possibly construct an encoding of $\prod_j a_j$ without using the same level twice. Using a variant of the multilinear Diffie-Hellman assumption, this scheme can be proved secure. This assumption can be justified in the generic multilinear map model, and hence our scheme can be proven secure in the weak CLT13 model.

Enforcing Matrices We believe our new enforcing matrices may be of interest beyond the immediate scope of this work, and we therefore briefly describe them. Consider a single-input vector-input branching program which reads the single input. Suppose the adversary is restricted to constructing linear combinations of iterated matrix products. We want to set up the matrices in a way so that the only linear combinations that give 0 correspond to sums of vector-input branching program evaluations.

We show that an equivalent condition is that the linear combinations are *permutation invariant*. This means the following. Let $\beta_{x_1,\ldots,x_\ell}$ be the coefficient of the linear combination corresponding to the iterated

matrix product that selects the x_ith matrix from the ith column. Then for any permutation σ on $[\ell]$, it must be that $\beta_{x_1,\ldots,x_\ell} = \beta_{x_{\sigma(1)},\ldots,x_{\sigma(\ell)}}$.

First, we consider enforcing just a single permutation σ. In this case, our enforcing matrices have a very simple form. For each row, choose random $\alpha_1, \ldots, \alpha_\ell$. The enforcing matrices for that row will have the form:

$$\begin{pmatrix} \alpha_1 & & & \\ & \alpha_{\sigma(1)} & & \\ & & \alpha_{\sigma^2(1)} & \\ & & & \ddots \end{pmatrix}, \begin{pmatrix} \alpha_2 & & & \\ & \alpha_{\sigma(2)} & & \\ & & \alpha_{\sigma^2(2)} & \\ & & & \ddots \end{pmatrix}, \begin{pmatrix} \alpha_3 & & & \\ & \alpha_{\sigma(3)} & & \\ & & \alpha_{\sigma^2(3)} & \\ & & & \ddots \end{pmatrix}, \ldots$$

Notice that adding the matrices from two different rows will still maintain a sequence of matrices with the above form. However, mixing and matching different subset-sums in each column will give matrices with a different structure. We therefore design bookend matrices to force the iterated matrix product of any row to be zero, while improper combinations will give non-zero. Next, we build full enforcing matrices by stitching together σ-enforcing matrices, where σ ranges over any set that generates the symmetric group S_n. For example, take σ to range over the transpositions $(1\ \ 2), (2\ \ 3), \ldots, (\ell - 1\ \ \ell)$. Notice that for transpositions, the enforcing matrices above repeat after the second diagonal entry. Therefore, it suffices to truncate the matrices to 2×2 matrices.

To create enforcing matrices for several-input vector-input branching programs, we build a set of enforcing matrices for each input. Then, we stitch these together as block-diagonal matrices. Roughly, each block corresponds to an input vector; in columns that read that input, we place the enforcing matrices, and in columns that do not, we place identity matrices.

Putting it all together, our multilinear maps are block diagonal, where the upper block is just a scalar representing the encoding, and the second block consists of enforcing matrices. Finally, we add a small 2×2 block to help with the analysis. The overall width of our matrices is therefore $2(\ell - 1)d + 3$, where d is the desired multilinearity and ℓ is the number of matrices per encoding.

2 Preliminaries

2.1 Multilinear Maps and the Generic Model

A multilinear map (also known as a graded encoding scheme) with universe set $\mathbb{U}$ and ring R supports encodings of plaintext elements in R at levels corresponding to subsets of $\mathbb{U}$. A plaintext element a encoded at $S \subseteq \mathbb{U}$ is denoted as $[a]_S$. Multilinear maps support some subset of the following operations on these encodings:

- (Encoding) Given an element $a \in R$ and level set $S \in \mathbb{U}$, output an encoding $[a]_S$.
- (Addition) Two encodings at the same level $S \subseteq \mathbb{U}$ can be added / subtracted. Informally,

$$[a_1]_S \pm [a_2]_S = [a_1 \pm a_2]_S.$$

- (Multiplication) An encoding at level $S_1 \subseteq \mathbb{U}$ can be multiplied with an encoding at level $S_2 \subseteq \mathbb{U}$, provided $S_1 \cap S_2 = \emptyset$. Their product is an encoding at level $S_1 \cup S_2$. Informally:

$$[a_1]_{S_1} \cdot [a_2]_{S_2} = [a_1 \cdot a_2]_{S_1 \cup S_2}.$$

- (Re-randomization) We will allow for schemes with non-unique encodings. In this case, we may want a re-randomization procedure, which takes as input an encoding $[a]_S$ of a potentially unknown element a, and outputs a "fresh" encoding of $[a]$, distributed statistically close to a directly encoding a.
- (Zero-Testing) An encoding $[a]_{\mathbb{U}}$ at level $\mathbb{U}$ can be tested for whether $a = 0$.
- (Extraction) An encoding $[a]_{\mathbb{U}}$ at level $\mathbb{U}$ can be extracted, obtaining a string r. Different encodings of the same a must yield the same r.

Most multilinear map schemes, due to security vulnerabilities, only support addition, multiplication, and zero-testing/extraction, but do not support public re-randomization or encoding. Instead, encoding must be performed by a secret key holder.

In the generic multilinear map model (also known as the standard ideal graded encoding model), a stateful oracle maintains a private table of plaintext elements along with their encoding levels. For each plaintext element, the oracle releases a corresponding public handle independent of the element's value. Any party can perform add/subtract/multiply operations on the public handles, as long as they respect the level structure specified above. The oracle performs the computation on the actual plaintexts, stores the result, and outputs a newly generated handle to this plaintext. Additionally, any party can request a zero-test on a public handle, in which case the oracle returns a bit indicating if the corresponding element is an encoding of zero at level $\mathbb{U}$ or not. We give a more precise description of the generic model in Section 4.

2.2 Overview of the CLT13 Multilinear Maps

We give a brief overview of the CLT13 multilinear maps, adapted from text in [CLLT16b]. For a full description of the scheme, see [CLT13]. The CLT13 scheme relies on the Chinese Remainder Theorem (CRT) representation. For large secret primes p_k, let $N = \prod_{k=1}^{n} p_k$. Let $\mathsf{CRT}(s_1, s_2, \ldots, s_n)$ or $\mathsf{CRT}(s_k)_k$ denote the number $s \in \mathbb{Z}_N$ such that $s \equiv s_k \pmod{p_k}$ for all $k \in [n]$. The plaintext space of the CLT13 scheme is $\mathbb{Z}_{g_1} \times \mathbb{Z}_{g_2} \times \cdots \times \mathbb{Z}_{g_n}$ for small secret integers g_k. An encoding of a vector $m = (m_1, \ldots, m_n)$ at level set $S = \{i_0\}$ is an integer $\alpha \in \mathbb{Z}_N$ such that $\alpha = \mathsf{CRT}(m_1 + g_1 r_1, \ldots, m_n + g_n r_n)/z_{i_0} \pmod{N}$ for small integers r_k, and where z_{i_0} is a secret mask in $\mathbb{Z}_N$ uniformly chosen during the parameters generation procedure of the multilinear map. To support κ-level multilinearity, κ distinct z_i's are used.

Additions between encodings in the same level set can be done by modular additions in $\mathbb{Z}_N$. Multiplication between encodings can be done by modular multiplication in $\mathbb{Z}_N$, only when those encodings are in disjoint level sets, and the resulting encoding level set is the union of the input level sets. At the top level set $[\kappa]$, an encoding can be tested for zero by multiplying it by the zero-test parameter $p_{zt} = \prod_{i=1}^{\kappa} z_i \cdot \mathsf{CRT}(p_k^* h_k g_k^{-1})_k \pmod{N}$ in $\mathbb{Z}_N$ where $p_k^* = N/p_k$, and comparing the result to N. If α encodes zero in each position, then it can be shown that

$$p_{zt}\alpha (\mathrm{mod} N) = \sum_{k=1}^{n} \gamma_k s_k$$

where γ_k are "smallish" global secret parameters that depend on the other CLT13 parameters, and s_k are the numerators of the CRT components of α: $\alpha = \frac{1}{\prod_{i=1}^{\kappa} z_i} \mathsf{CRT}(s_k)_k$. These s_k are "small", so for encodings of zero, $p_{zt}\alpha (\bmod N)$ is small relative to N. If α does not encode 0, then one heuristically expects $p_{zt}\alpha (\bmod N)$ to be large relative to N[6]. Thus, we can zero test by determining if this quantity is small. We can also extract a unique representation of any encoded element by computing $p_{zt}\alpha (\mathrm{mod} N)$, and rounding appropriately.

2.3 Schwartz-Zippel Variant

The standard Schwartz-Zippel Lemma applies to uniformly and independently sampled random variables. However, we will need to consider polynomials over CLT13 encodings as well as terms that appear in the zero-testing parameter. These can be seen as random variables, but they are neither independent nor uniformly sampled. Fortunately, Garg et al. give a variant of Schwartz-Zippel in [GMM+16] that applies to arbitrarily correlated random variables, as long as each variable has sufficient min-entropy conditioned on the other variables. We state the lemma here with minor modifications.

Let $P \in \mathbb{Q}[x_1, \ldots, x_n]$ be an arbitrary polynomial of degree at most d. Let $X_1, \ldots, X_n$ be potentially correlated random variables over $\mathbb{Q}$. For each X_i, let $S(X_i)$ be the set of values $x_i \in \mathbb{Q}$ such that $\Pr[X_i =$

[6] In the full CLT13, there is a vector of zero-testing elements created in a way to prove that the result is large for non-zero encodings. However, in practice this is much less efficient, so most implementations only use a single zero test vector as describe here.

$x_i] > 0$. We assume that for each X_i, $S(X_i)$ is finite. Let $p_i(x_1, \ldots, x_{i-1})$ be the guessing probability of X_i conditioned on $X_j = x_j$ for each $j < i$. So

$$p_i(x_1, \ldots, x_{i-1}) = \max_{x_i \in S(X_i)} \Pr[X_i = x_i | X_j = x_j \forall j < i]$$

Let p_i be the expectation of $p_i(x_1, \ldots, x_{i-1})$ when x_j are drawn from $X_j : p_i = E[p_i(X_1, \ldots, X_{i-1})]$. Let $p_{\max} = \max_i p_i$ be the maximum of the p_i.

Lemma 1. *Let $d, n, P, X_1, \ldots, X_n, p_{\max}$ be as above. Then*

$$\Pr_{X_1,\ldots,X_n}[P(X_1, \ldots, X_n) = 0] \leq d \cdot p_{\max}.$$

For the proof of this lemma, see [GMM+16]. We note that the original statement of this lemma in [GMM+16] uses a finite field instead of $\mathbb{Q}$, but the proof easily extends to the infinite case.

2.4 Matrix Branching Programs

Intuitively, a single-input matrix branching program consists of a sequence of pairs of matrices, a pair of fixed bookend vectors, and an input selection function. To evaluate the program on an input bit-string, one matrix is chosen from each pair by reading a single bit of the input, as determined by the input selection function. The matrices are multiplied together in order, and the resulting product matrix is left and right multiplied by the bookend vectors, resulting in a scalar value. We will sometimes use this scalar as the output of the branching program, and other times use the result of testing whether this scalar is zero. This notion can be extended to the dual-input setting, where two input bits are read at each time, and thus one of four matrices is chosen at each step of the sequence.

We roughly follow the generalized matrix branching program definition of Badrinarayanan et al. [BMSZ16], which allows for matrices of arbitrary dimensions, so long as the resulting product is always a scalar. However, we deviate from their definition slightly by using fixed left and right bookend vectors.

Definition 1. *A single-input branching program of length ℓ and shape $(d_0, d_1, \ldots, d_\ell) \in (\mathbb{Z}^+)^{\ell+1}$ for n-bit inputs is given by*

$$BP = \left(\mathsf{inp}, s, t, \{\boldsymbol{B}_{i,0}, \boldsymbol{B}_{i,1}\}_{i \in [\ell]}\right)$$

where $\boldsymbol{B}_{i,b} \in \mathbb{Z}^{d_{i-1} \times d_i}$ are $d_{i-1} \times d_i$ matrices, s is a d_0-dimensional row vector, t is a d_ℓ-dimensional column vector, and $\mathsf{inp} : [\ell] \to [n]$ is the evaluation function of BP. $BP : \{0,1\}^n \to \mathbb{Z}$ is computed as

$$BP(x) = s \cdot \left(\prod_{i=1}^{n} \boldsymbol{B}_{i, x_{\mathsf{inp}(i)}}\right) \cdot t$$

2.5 Vector-Input Branching Programs

We generalize matrix branching programs to vector-input branching programs, a new notion we define. With single-input branching programs, each set in the sequence of matrices contains two matrices, and an input bit selects one of them. Our vector input generalization allows for selecting a *linear combination* of the matrices in any set. Each set now contains k matrices, and thus program inputs consist of n non-zero vectors of dimension k.

For this paper, we will only need to consider a restricted class of such programs. Specifically, we use read-t programs, meaning that each vector in the input is read exactly t times. These programs are single-input, so there are nt sets of matrices. Furthermore, the input selection is fixed so that the ith input vector is read for matrix sets $i, n+i, \ldots, (t-1)n+i$. In other words, the input selection function is simply $\mathsf{inp}(j) = j \pmod n$. Each of the n input vectors is restricted to have non-negative integer components that sum to t. Let $H_{\ell,k}$ denote the set of all such k-dimensional vectors.

Definition 2. *A read-t vector-input branching program with input length n, vector dimension k, and shape* $(d_0, d_1, \ldots, d_{n\ell}) \in (\mathbb{Z}^+)^{n\ell+1}$ *is given by a sequence*

$$VBP = \left(s, t, \{\boldsymbol{B}_{i,j}\}_{i\in[n\ell], j\in[k]}\right)$$

where $\boldsymbol{B}_{i,j} \in \mathbb{Z}^{d_{i-1}\times d_i}$ *are* $d_{i-1} \times d_i$ *matrices,* s *is a* d_0*-dimensional row vector, and* t *is a* $d_{n\ell}$*-dimensional column vector.* $VBP : H^n_{k,\ell} \to \mathbb{Z}$ *is computed as*

$$VBP(x) = s \cdot \left(\prod_{i=1}^{n\ell}\left(\sum_{j=1}^{k} x_{(i \bmod n),j}\boldsymbol{B}_{i,j}\right)\right) \cdot t$$

2.6 Kilian Randomization of Matrix Sequences

Consider a collection of n columns of matrices, where each column may contain an arbitrary polynomial number of matrices. Denote the jth matrix in column i as $A_{i,j}$. Suppose the matrices within each column have the same dimensions, and across columns have compatible dimensions so that matrices in adjacent columns can be multiplied together, and multiplying one matrix from each column results in a scalar. Kilian [Kil88] describes a method to partially randomize such branching programs. Random square matrices R_i are chosen at random. Then matrix $A_{i,j}$ is left-multiplied by R_i^{-1} and right-multiplied by R_{i+1}. When performing an iterated matrix product selecting one matrix from each column, the R_i and R_i^{-1} cancel out, so the product is unchanged by this randomization.

3 A New Lemma

In this section, we state a generalization of a lemma by Badrinarayanan et al. [BMSZ16]. Consider the same collection of matrices as in the preceding section. We have a set S (possibly of exponential size) of n-tuples of indices that define a set of allowable iterated matrix products. For example, if $n = 3$ and S contains the tuple $(7, 4, 8)$, this allows for multiplying matrix 7 in column 1 by matrix 4 in column 2 by matrix 8 in column 3. We enforce that all matrices in the same column share the same dimensions.

We let $X = \{X_1, \ldots, X_m\}$ denote a set of variables, and we enforce that every matrix element is a polynomial over these variables. For a set of vectors $V = \{v_1, \ldots, v_\ell\}$ whose entries are polynomials over X, we say that V is *linearly independent* if there is no sequence of *constants* $a_1, \ldots, a_\ell$, not all of which are 0, such that $\sum_i a_i v_i$ is the identically 0 vector. Note that this notion of linear independence allows for arbitrarily-many linearly independent vectors in a finite-dimensional space. For example, $1, X, X^2, \cdots$ are all linearly independent according to this notion.

Definition 3. *An allowable polynomial over* S *is a (possibly exponentially sized) polynomial where each monomial is a product of exactly one matrix entry from each column of matrices, subject to the condition that the matrices the entries are drawn from are an explicitly allowed product in* S*.*

One example of an allowable polynomial is an *allowable matrix product*, which picks one tuple from S, and multiplies all the matrices corresponding to that tuple:

Definition 4. *An allowable matrix product over* S *is the iterated matrix product corresponding to some tuple* $t \in S$*.*

We can apply Kilian randomization to these matrices. We treat each entry of the Kilian randomization matrices as distinct formal variables, and denote this set of variables by R. Let $\widehat{A_{i,j}}$ denote the matrix over variables in X and R corresponding to $A_{i,j}$ after randomization.

Lemma 2. *Suppose a collection of matrices* $\{A_{i,j}\}$ *of polynomials over the variables* X *along with a set* S *of valid products satisfy the following conditions:*

– *(Left non-shortcutting) For each member of S, multiply every corresponding matrix except for the rightmost matrix. Interpret the resulting matrices as vectors of polynomials over the variables X. These vectors must be linearly independent.*
– *(Right non-shortcutting) Defined analogously to left non-shortcutting, except products do not include the leftmost matrix. These vectors must be linearly independent in the sense above.*

Any allowable polynomial over the $\widehat{A_{i,j}}$ matrices that is identically zero over the formal variables in X and R can be written as a linear combination of allowable matrix products over S. Additionally, this polynomial evaluated over the non-randomized $A_{i,j}$ matrices, is identically zero over the formal variables X.

The proof of this lemma is given in A.

4 The Model

In this section, we define two models. The first is the weak CLT13 model (Section 4.1), intended to capture all known classical attacks on the CLT13 multilinear maps. The second is the CLT13 plain annihilation model, a modification of the weak CLT13 model that changes the winning conditions (Section 4.2). We justify our first model by proving that it captures known attacks from the literature. While the weak CLT13 model captures all known attacks, it can be a bit cumbersome to use. Our second "plain annihilation model" is designed to be a much simplified attack model to make it easier to prove security. The main theorem of this section is that an adversary in the weak CLT13 model implies the existence of an adversary in the CLT13 annihilation model. To demonstrate the ease of proving security in the simplified model, we show that combining this theorem with the Branching Program Un-Annihilatability Assumption of [GMM+16] immediately shows the virtual black box (VBB) security of the obfuscator of Badrinarayanan et al. [BMSZ16]. Additionally, this shows that the order-revealing encryption scheme of Boneh et al. [BLR+15] is secure in our model.

4.1 CLT13 Weak Multilinear Map Model

Notation. We will let uppercase letters such as M, S, Γ denote formal variables, and lower case letters such as m, s, γ denote actual values. Let m_{ji} be a set of elements indexed by j and i. We introduce $\langle \mathbf{m} \rangle_i$ as shorthand for the set $\{m_{ji}\}_j$ of all elements with index i, and $\langle \mathbf{m} \rangle$ to denote the set $\{\langle \mathbf{m} \rangle_i\}_i$. For a set $M_{j,i}$ of formal variables indexed by j and i, define $\langle \mathbf{M} \rangle_i$ and $\langle \mathbf{M} \rangle$ analogously.

With this notation, we now define our weak CLT13 model. The model consists of the following interfaces:

Initialize parameters. At the beginning of the interaction with the model $\mathcal{M}$, $\mathcal{M}$ is initialized with the security parameter λ and the multilinearity parameter $\kappa \leq \mathsf{poly}(\lambda)$. We generate the necessary parameters of the CLT13 scheme (including the vector dimension n, the primes g_i, p_i for $i \in [n]$) according to the distributions suggested by Coron et al. [CLT13]. Let $R_{\mathsf{ptxt}} = \mathbb{Z}_{\prod_i g_i} = \otimes_i \mathbb{Z}_{g_i}$ be the plaintext ring. Let $R_{\mathsf{ctxt}} = \mathbb{Z}_{\prod_i p_i} = \otimes_i \mathbb{Z}_{p_i}$. We will usually interpret elements in R_{ptxt} and R_{ctxt} as vectors of their Chinese Remaindering components.

Initialize elements. Next, $\mathcal{M}$ is given a number of plaintext vectors $\mathbf{m}_j \in R$ as well as an encoding level S_j for each plaintext. $\mathcal{M}$ generates the CLT13 numerators $\mathbf{s}_j$ where $s_{ji} = m_{ji} + g_i r_{ji}$ as in the CLT13 encoding procedure. For each j, $\mathcal{M}$ stores the tuple $(\mathbf{m}_j, \mathbf{s}_j, S_j)$ in the a pre-zero test table.

Zero-testing. The adversary submits a polynomial p_u to $\mathcal{M}$, represented as a polynomial-sized *level-respecting* algebraic circuit. Here, level-respecting means that all wires are associated with a level S, input wires are associated to the sets S_j, add gates must add wires with the same level and output a wire with the same level, multiply gates must multiply wires with sets $S_0 \cap S_1 = \emptyset$ and output a wire with the level $S_0 \cup S_1$, and the final output wire must have set $\{1, \ldots, \kappa\}$.

Next, $\mathcal{M}$ checks whether $p_u(\langle \mathbf{m} \rangle_i) = 0$ for all i, or equivalently whether p_u evaluates to 0 over R when applied to the plaintext elements. If the check fails, $\mathcal{M}$ returns 1. If this check passes, then $\mathcal{M}$ returns 1.

We assume without loss of generality that the set $\{p_u\}$ of *successful* zero tests are linearly independent as polynomials (since otherwise a zero-test on one p_u can be derived from the result of a zero-test on several other p_u).

If we stop here, we recover the plain generic multilinear map model [BR14]. However, in our model, a successful zero test does more. If zero testing is successful, p_u corresponds to a valid construction of a top-level zero encoding. $\mathcal{M}$ then additionally returns a handle T_u to the value $t_u = \sum_i \gamma'_i p_u(\langle \mathbf{s} \rangle_i)$, the result of the zero-test computation. Each handle T_u as well as the zero-test result is stored in a *zero-test table*.

Post-zero-test. Finally, the adversary submits a polynomial Q on the handles $\{T_u\}_u$ and the formal variables $\langle \mathbf{S} \rangle_i$ for some $i \in [n]$ (that $\mathcal{A}$ picks). The model looks up each handle T_u in the zero-test table and plugs in the corresponding values t_u. The model outputs "WIN" if the following two conditions are satisfied.

1. $Q(\{t_u\}_u, \langle \mathbf{S} \rangle_i) \not\equiv 0$ as a polynomial over the formal variables $\langle \mathbf{S} \rangle_i$.
2. $Q(\{t_u(\langle \boldsymbol{\gamma}' \rangle, \langle \mathbf{s} \rangle)\}_u, \langle \mathbf{s} \rangle_i) = 0 \bmod p_i$.

These two conditions imply that Q is a polynomial with non-zero degree over the $\langle \mathbf{S} \rangle_i$ formal variables that can be solved for the values $\langle \mathbf{s} \rangle_i$.

4.2 CLT13 Plain Annihilation Model

We define a simplification of the above CLT13 weak multilinear map model, which is identical except for post-zero-test queries:

Post-zero-test. $\mathcal{A}$ submits a polynomial Q' on a set of formal variables $\{P_u\}_u$, where P_u represents the successful zero test polynomials p_u. The model outputs "WIN" if the following conditions are satisfied.

1. $Q(\{P_u\}_u)$ *is not* identically zero over its formal variables
2. $Q(\{p_u(\langle \mathbf{S} \rangle_i\}_u)$ *is* identically zero over the $\langle \mathbf{S} \rangle_i$ formal variables.

In other words, $\mathcal{A}$ wins if it submits an annihilating polynomial for the set of $\{P_u\}_u$ polynomials.

4.3 Classical Attacks in the Weak CLT13 Model

In this section, we show that the original attack on the CLT13 multilinear maps by Cheon et al. fits into our framework [CHL+15]. Then, we show that the more general attack strategy of Coron et al. can also be expressed in our model [CGH+15].

Cheon et al. Attack. Mounting this attack requires that the set of plaintext vectors $\{\mathbf{m}_j\}$ given to M can be divided into three distinct sets of vectors, A, B, C that satisfy certain properties. We can discard/ignore any other plaintext vectors. For ease of exposition, we relabel the vectors in these sets as:

$$A = \{\mathbf{m}_1^A, \ldots, \mathbf{m}_n^A\} \qquad B = \{\mathbf{m}_1^B, \mathbf{m}_2^B\} \qquad C = \{\mathbf{m}_1^C, \ldots, \mathbf{m}_n^C\}$$

These vectors can be encoded at arbitrary levels, as long as for any j, σ, k, $\mathbf{m}_j^A \cdot \mathbf{m}_\sigma^B \cdot \mathbf{m}_k^C$ is a plaintext of zeros at the top level. Accordingly, $\mathcal{A}$ submits polynomials $p_{j,\sigma,k}$ for all $j, k \in [n], \sigma \in [2]$ for zero-testing where

$$p_{j,\sigma,k}(m_1^A, \ldots, m_n^A, m_1^B, m_2^B, m_1^C, \ldots, m_n^C) = m_j^A \cdot m_\sigma^B \cdot m_k^C$$

Each of these polynomials clearly gives a successful zero-test. In response to each query, $\mathcal{M}$ returns a handle $T_{j,\sigma,k}$ to the value

$$t_{j,\sigma,k} = \sum_{i=1}^{n} \gamma_i s_{j,i}^A \cdot s_{\sigma,i}^B \cdot s_{k,i}^C.$$

For $\sigma \in \{1, 2\}$, define W_σ to be the $n \times n$ matrix whose (j, k)th entry is $T_{j,\sigma,k}$. In the real attack, the adversary computes the matrix $W_1 W_2^{-1}$, which Cheon et al. [CHL+15] show has eigenvalues $\frac{s_{1,i}^B}{s_{2,i}^B}$. The

adversary solves the characteristic polynomial of $W_1 W_2^{-1}$ for these eigenvalues. In our model $\mathcal{A}$ cannot immediately submit this characteristic polynomial, as it involves rational functions of the handles T, and can only be solved for ratios of the s_{ji} values. However, we observe that the characteristic polynomial

$$\det\left(W_1 W_2^{-1} - \lambda I\right) = \det\left(W_1 W_2^{-1} - \left(\frac{S^B_{1,i}}{S^B_{2,i}}\right) I\right) = 0$$

can be re-written by substituting $W_2^{-1} = \frac{W_2^{adj}}{\det(W_2)}$. (where W_2^{adj} denotes the adjoint matrix of W_2). Applying properties of the determinant then gives

$$\det(W_1 W_2^{adj} S^B_{2,i} - S^B_{1,i} \det(W_2) I) = 0$$

$\mathcal{A}$ submits the left-hand side expression above as its polynomial Q in a post-zero-test query. Since the Cheon et al. attack is successful, we know Q is nonzero over the formal variables $\langle \mathbf{S} \rangle_i$ after the values associated with the handles T are plugged in. Additionally, plugging in the appropriate solutions $\langle \mathbf{s} \rangle_i$ satisfies the above expression, so both win conditions are satisfied. Thus, $\mathcal{A}$ wins in our model.

Coron et al. General Attack. We consider the general attack strategy of [CGH+15] on CLT13-based schemes, and show that these attacks fit within our model. Note that these attacks are on matrices of CLT13 encodings, and it is cumbersome to express these attacks in terms of the notation we use for our weak CLT13 model. For this section, we largely adopt the notation of [CLLT17], but explain how each step of their attack works in our model.

Coron et al. [CGH+15] define an attack set of dimension d as consisting of three sets of matrices $A, B, C \subset \mathbb{Z}_N^{d \times d}$ of size $|A| = |C| = nd$ (where n is the number of CLT13 secret primes and $N = \prod_{i=1}^n p_i$) and $|B| = 2$ and two vectors $s \in \mathbb{Z}_N^{1 \times d}$ and $t \in \mathbb{Z}_N^{d \times 1}$. These sets are such that

$$W_\sigma[j,k] := s \times A_j \times B_\sigma \times C_k \times t \in \mathbb{Z}_N$$

is a zero-tested top level encoding of 0 for each choice of $j, k \in [n] \times [n]$ and $\sigma \in \{0,1\}$. Here $[z]_p$ will denote the unique integer in $[-p/2, p/2)$ congruent to z mod p, and this notation extends entry-wise to vectors and matrices. Naturally, this gives rise to two matrices W_σ for $\sigma \in \{0,1\}$, where the (j,k) entry is given by $W_\sigma[j,k]$.

In [CGH+15], an attack set is called "good" if it can be used to mount their attack on CLT13-based schemes. One property of a good attack set is that

$$\mathrm{char}(W_0 \times W_1^{-1}) = \prod_{j \le n} \mathrm{char}([B_0]_{p_j} \times [B_1]^{-1}_{p_j}),$$

where $\mathrm{char}(W)$ denotes the characteristic polynomial of matrix W.

Given an attack set $\{A_j\}_j, \{B_\sigma\}_\sigma, \{C_k\}_k, s, t$, the adversary first computes the matrices W_0, W_1 where $W_\sigma[j,k] := [s \times A_j \times B_\sigma \times C_k \times t]_N$.

The (j,k) entry of the W_σ matrix is the result of a zero test on the polynomial

$$\sum_{\ell', k', j', i'} s_{i'} \cdot A_j[i', j'] \cdot B_\sigma[j', k'] \cdot C_k[k', \ell'] \cdot t_{\ell'}.$$

Thus, an adversary $\mathcal{A}$ in our model can obtain handles to all elements of the matrices W_σ.

Let B_j be the matrix $[B_0]_{p_j} \cdot [B_1]^{-1}_{p_j}$. Note that $char(B_j)$ divides $char(W)$ since A, B, C, s, t is a good attack set. Furthermore, B_j satisfies its own characteristic equation, so we conclude that it satisfies the characteristic equation of W. This fact can be expressed as

$$\det(W_0 \times W_1^{-1} - [B_0]_{p_j} \times [B_1]^{-1}_{p_j}) = 0,$$

which we can rewrite as

$$\det(\ \det([B_1]_{p_j}) \cdot W_0 \times W_1^{adj} - \det(W_1) \cdot [B_0]_{p_j} \times [B_1]^{adj}_{p_j}\ ) = 0$$

An adversary $\mathcal{A}$ in our model can express the above polynomial as a post-zero-test query polynomial. Simply view the matrices W_0, W_1 as matrices of handles $\mathbf{T}$ given in response to zero tests, while treating the entries of B_0, B_1 as a subset of the formal variables $\langle S \rangle_i$.

4.4 Model Conversion Theorem

Theorem 1. *If there exists an adversary $\mathcal{A}$ that wins with non-negligible probability in the weak CLT13 multilinear map model, there exists an adversary $\mathcal{A}'$ that wins with non-negligible probability in the CLT13 annihilation model. $\mathcal{A}'$ is the same as $\mathcal{A}$ up to and including the zero-test queries, and only differs on the post-zero test queries.*

Proof. Suppose $\mathcal{A}$ produces a post-zero-test Q such that $Q(\{t_u\}_u, \langle \mathbf{S} \rangle_i) \neq 0$ and $Q(\{t_u\}_u, \langle \mathbf{s} \rangle_i) = 0$. Recall that t_u is a polynomial over $\langle \mathbf{s} \rangle$: $t_u(\langle \boldsymbol{\gamma} \rangle, \langle \mathbf{s} \rangle) = \sum_i \gamma_i p_u(\langle \mathbf{s} \rangle_i)$. Note that the first condition above implies that $Q(\{t_u(\langle \boldsymbol{\gamma} \rangle, \langle \mathbf{s} \rangle)\}_u, \langle \mathbf{S} \rangle_i)$ is not identically zero as a polynomial in the formal variables $\boldsymbol{\Gamma}, \langle \mathbf{S} \rangle$, where we have hard-coded the actual values of t_u into the first set of inputs to Q. We claim that $Q(\{t_u(\langle \boldsymbol{\Gamma} \rangle, \langle \mathbf{S} \rangle)\}_u, \langle \mathbf{S} \rangle_i)$ must be identically 0, where we have substituted the definition of t_u, but left $\langle \mathbf{S} \rangle$ and $\boldsymbol{\Gamma}$ in the definition of t_u as formal variables.

Suppose Q was not identically zero over $\boldsymbol{\Gamma}, \langle \mathbf{S} \rangle$. Then with overwhelming probability Q would not have evaluated to zero, and hence would not have given a CLT13 break. This follows from applying a variant of the Schwartz-Zippel lemma due to Garg et al. [GMM+16] (stated in Section 2.3 as Lemma 1), treating the s_{ji} and γ_i as the random variables, and showing that $p_{\max}$ is negligible. Since $s_{ji} = a_{ji} + g_i r_{ji}$ and each r_{ji} is sampled independently from a high entropy distribution, the s_{ji}'s have significant min-entropy conditioned on the other s_{ji} and γ_i variables. The γ_i variables have significant min-entropy since $\gamma_i = \frac{h_i}{x_0} p_i g_i$, and the h_i's are random β-bit integers.

Next, consider expanding out the polynomial $Q(\{T_u\}_u, \langle \mathbf{S} \rangle_i)$ in terms of its formal variables. We can group together all the T_u terms of each degree. When substituting $T_u = t_u(\langle \boldsymbol{\Gamma} \rangle, \langle \mathbf{S} \rangle)$, all the T_u terms of the same degree will have the same degree for the $\Gamma_{i'}$ variables, and thus these terms can only be canceled out by other T_u terms of that degree. So we will restrict our attention to the case where all the t_u terms have some fixed degree d_t, since we can just derive such a polynomial from the given Q. Thus, the polynomial $Q(\{t_u(\langle \boldsymbol{\Gamma} \rangle, \langle \mathbf{S} \rangle)\}_u, \langle \mathbf{S} \rangle_i)$ can be written in the form $\sum_{|S|=d_t} a_S t_{S_1} t_{S_2} \cdots t_{S_{d_t}} R_S(\langle \mathbf{S} \rangle_i)$, where S runs over all multi-sets of size d_t of elements from $[U]$ (where $[U]$ is the set of indices $\{1, \ldots, U\}$ of polynomials P_u submitted by the adversary), and we denote the elements of S as $S_1, S_2, \ldots, S_{d_t}$. This can be expanded as

$$\sum_{|S|=d_t} a_S \sum_{i'} \Gamma_{i'} p_{S_1}(\langle \mathbf{S} \rangle_{i'}))(\sum_{i'} \Gamma_{i'} p_{S_2}(\langle \mathbf{S} \rangle_{i'})) \cdots (\sum_{i'} \Gamma_{i'} p_{S_{d_t}}(\langle \mathbf{S} \rangle_{i'})) R_S(\langle \mathbf{S} \rangle_i),$$

and in particular if we look at the coefficient of $\Gamma_k^{d_t}$ for any $k \neq i$, it is

$$\sum_{|S|=d_t} a_S p_{S_1}(\langle \mathbf{S} \rangle_k) p_{S_2}(\langle \mathbf{S} \rangle_k) \cdots p_{S_{d_t}}(\langle \mathbf{S} \rangle_k) R_S(\langle \mathbf{S} \rangle_i),$$

which we notice is equivalent to $Q(\{p_u(\langle \mathbf{S} \rangle_k)\}_u, \langle \mathbf{S} \rangle_i)$. Since $Q(\{t_u(\langle \boldsymbol{\Gamma} \rangle, \langle \mathbf{S} \rangle)\}_u, \langle \mathbf{S} \rangle_i) \equiv 0$, in particular the coefficient of $\Gamma_k^{d_t}$ should be 0 (since there is nothing else for the $\Gamma_k^{d_t}$ term to cancel out with), so we have $Q(\{p_u(\langle \mathbf{S} \rangle_k)\}_u, \langle \mathbf{S} \rangle_i) \equiv 0$.

If we think of the S_{ji} terms as the variables of this polynomial and everything else as comprising the coefficients, we note that all the coefficients must be identically zero, since the S_{jk} formal variables are disjoint from the S_{ji} formal variables. Write this polynomial as $\sum_V c_V(\{p_u(\langle \mathbf{S} \rangle_k)\}_u) \prod_{j \in V} S_{ji}$ where V scrolls over multisets of indices j, meaning $c_V(\{p_u(\langle \mathbf{S} \rangle_k)\}_u) \equiv 0$ for all V.

Recall that $Q(\{t_u(\langle \boldsymbol{\gamma} \rangle, \langle \mathbf{s} \rangle)\}_u, \langle \mathbf{S} \rangle_i) \not\equiv 0$, which implies that $Q(\langle \mathbf{T} \rangle, \langle \mathbf{S} \rangle_i) \not\equiv 0$, where $\langle \mathbf{T} \rangle$ denotes the set $\{T_u\}_u$ of formal variables corresponding to the original $\{t_u\}_u$ polynomials. We can write this polynomial as $Q(\langle \mathbf{T} \rangle, \langle \mathbf{S} \rangle_i) = \sum_V c_V(\langle \mathbf{T} \rangle) \prod_{j \in V} S_{ji}$ where V scrolls over multisets of indices j. The fact that this polynomial is not identically zero implies that there exists a V such that $c_V(\langle \mathbf{T} \rangle)$ is not identically zero.

Renaming the $\langle \mathbf{T} \rangle$ variables as $\langle \mathbf{P} \rangle$ variables, we therefore have that for some V, $c_V(\langle \mathbf{P} \rangle)$ is a non-identically zero polynomial such that $c_V(\{p_u(\langle \mathbf{S} \rangle_k)\}_u) \equiv 0$, but this means it is an annihilating polynomial for the $\{p_u(\{S_{ji}\})\}_u$ polynomials.

4.5 Secure Obfuscation and Order-Revealing Encryption in the Weak CLT13 Model

Security in our weak CLT13 model means security in the plain generic multilinear map model, plus the inability to construct an annihilating polynomial. We observe that Badrinarayanan et al. [BMSZ16] show for their obfuscator (which is a tweak of the obfuscator of Barak et al. [BGK+14]), the only successful zero tests an adversary can perform are linear combinations of honest obfuscation evaluations on some inputs. Moreover, the linear combinations can only have polynomial support. Recall that in [BMSZ16], evaluation is just a branching program evaluation over the encoded values. Therefore, any annihilating polynomial in the plain annihilation model is actually an annihilating polynomial for branching programs. Therefore, using the branching program un-annihilatability assumption of Garg et al. [GMM+16], we immediately conclude that no such annihilating polynomial is possible. Thus, there is no weak CLT13 attack on this obfuscator.

We similarly observe that in the order-revealing encryption (ORE) scheme of Boneh et al. [BLR+15], any successful zero is also a linear combination of polynomially-many branching program evaluations. Therefore, by a similar argument, we immediately obtain that Boneh et al.'s scheme is secure in our weak CLT13 model.

5 Our Multilinear Map Candidate

In this section, we give a candidate constant-degree multilinear map scheme. We show, given an assumption about annihilating vector-input branching programs, that this multilinear map is secure in the weak CLT13 model.

5.1 Construction Overview

The levels will be non-empty subsets of $[d]$ for some *constant* d. For simplicity, here we describe how to build a multilinear map that only allows a pairing operation that takes d elements, one from each singleton set, directly to a top-level encoding. This is the style of multilinear map envisioned by Boneh and Silverberg [BS03]. In Section 5.4, we explain how to extend our multilinear map to allow intermediate levels.

Our construction is logically organized into $d\ell$ columns, numbered from 1 to $d\ell$. The columns are further partitioned into d groups numbered 1 through d of ℓ columns, where the columns in each group are interleaved: group u consists of columns $u, u+\ell, u+2\ell$, etc. Each column will correspond to one level of the underlying CLT13 maps, and each group of columns will correspond to one meta-level of our scheme.

We first describe the format of a meta-encoding in our scheme. An encoding at singleton level u will consist of ℓ matrices of CLT13 encodings, one in each of the columns corresponding to column group u. We will denote by A_i the ith matrix in the encoding. A_i is constructed by first defining the diagonal matrix $\widetilde{A_i}$, of the form

$$\mathrm{diag}(m_i, v_i, w_i, \xi_1 I, \ldots, \xi_{i-1} I, E_i, \xi_{i+1} I, \ldots, \xi_d I).$$

These components of the diagonal matrix work as follows:

- m_i is a plaintext element used for the actual plaintext encoding.
- v_i and w_i are elements freshly sampled at random for this matrix. Their sole purpose is to enforce a requirement called non-shortcutting, which will arise in the security proof. They are canceled out in valid products by 0's in the bookend vectors, defined later.
- The remainder of the diagonal consists of ℓ block matrices, where $\ell - 1$ of them are set to random multiples of the identity, and the uth matrix is set to be an "enforcing matrix" E_i. The purpose of these matrices is roughly to prevent an adversary from arbitrarily mixing and matching the matrices from different encodings. For the sake of clarity, we defer the details of these matrices to Section 5.2.

Next, $d\ell + 1$ Kilian randomization matrices R_i are generated [Kil88], once for each column (plus an additional matrix). All encodings will share the same Kilian matrices. The $\widehat{A_i}$ matrices are left- and right-multiplied by the appropriate Kilian matrices, giving

$$\widehat{B_i} = R^{-1}_{u+(i-1)d-1} \widetilde{A_i} R_{u+(i-1)d}.$$

Each element of this matrix is encoded in an asymmetric CLT13 multilinear map at level $\{u + (i-1)d\}_{CLT13}$ (we differentiate levels of the underlying CLT13 map with this subscript, to avoid confusion with the levels of our multilinear map) corresponding to the column it belongs to. The resulting matrix of CLT13 encodings is taken to be A_i.

For a matrix A_i, we refer to the underlying m_i as the *matrix plaintext*. This is the only component of the matrix used for encoding actual plaintext elements. Therefore, as described so far, every meta-encoding in our scheme encodes a length-ℓ vector of ring elements.

We also give out "bookends" s, t, which are CLT13 encodings of the vectors

$$\hat{s} = (1, 1, 0, F_1, \ldots, F_d) \cdot R_0 \ , \ \hat{t} = R^{-1}_{d\ell} \cdot (1, 0, 1, G_1, \ldots, G_d)^T \ .$$

For the sake of clarity, we defer discussing the F, G vectors until Section 5.2. The 1 in the first position is used to extract the matrix plaintexts. The 0 in the second position of $\hat{t}$ will zero-out the v_i terms, while the 0 in the third position of $\hat{s}$ will zero-out the w_i terms. s is encoded at CLT13 level 0, while t is encoded at level $d\ell + 1$.

Let R_{ptxt} be the CLT13 plaintext ring. A meta-encoding of $x \in R_{\mathsf{ptxt}}$ at singleton level $\{u\}$ is simply a sequence of matrices $(A_i)_{i\in[\ell]}$ whose corresponding sequence of matrix plaintexts is $(x, 0, 0, \ldots, 0)$.
At instance generation, we generate and publish a set of initial public encodings.

- For each singleton level $\{u\} \subseteq [d]$, we publish encodings of $1, 2, 4, \ldots, 2^{\rho-1}$, where ρ is specified later.
- For each singleton level $\{u\} \subseteq [d]$, we publish τ encodings of zero, where τ is specified later.
- For the top level $[d]$, we publish a special *pre-zero-test encoding* that will have most of the structure of a valid top level encoding, except that it will not correctly encode an actual plaintext element. Its sequence of matrix plaintexts will be $(0, 1, 1, \ldots, 1)$, which differs from a normal encoding where the matrix plaintexts are all 0 after the first slot. The sole purpose of this encoding is to be added to any top level encoding we seek to zero test. Roughly, the element submitted for zero testing is the product of an encoding's matrix plaintexts, and without this step the product would always be zero.

To add/subtract two meta-encodings at the same singleton level $\{u\}$, which are two sequences of ℓ matrices, we line up the sequences of matrices and add/subtract the corresponding matrices component-wise. The resulting sequence of ℓ matrices is taken as the encoding of the sum. Intuitively, this works because adding these matrices also adds the sequence of matrix plaintexts. As we show in Section 5.2, the structure of the enforcing matrices is also preserved. If the input encodings have matrix plaintexts $(x_1, 0, \ldots, 0)$ and $(x_2, 0, \ldots, 0)$, the result of addition/subtraction has matrix plaintexts $(x_1 \pm x_2, 0, \ldots, 0)$.

To pair d meta-encodings, one from each singleton level, we do the following. For each $i \in [\ell]$, we line up the ith matrix from each encoding, in the order specified by the columns of our scheme, and multiply the matrices together. The resulting i matrices are then added to the corresponding i matrices from the pre-zero-test encoding. Based on the structure of our encoding scheme, the resulting i matrices have the matrix plaintext sequence $(\prod_u x_u, 1, \ldots, 1)$, where x_u was the value encoded at level $\{u\}$. Finally, we multiply all of these matrices together. The resulting matrix will have $\prod_u x_u$ in the upper-left corner. Finally, we multiply by the bookends s, t to obtain a single top-level CLT13 encoding. We set up the enforcing matrices in Section 5.2 to guarantee that this product becomes a CLT13 encoding of $\prod_u x_u$.

The remaining procedures work as follows.

Encode. To encode a plaintext $x \in \mathbb{Z}_{2^\rho}$ at a singleton level $\{u\}$, write the plaintext in base 2, and then sum the appropriate public encodings of powers of 2. We note that we do not publish a description of the plaintext ring $R_{\mathsf{ptxt}} = \mathbb{Z}_M$, where $M = \prod_i g_i$. Therefore, the input to the encoding procedure is some integer

$x \in \mathbb{Z}_{2^\rho}$, and the output is an encoding of $x \bmod M$, where $R_{\mathsf{ptxt}} = \mathbb{Z}_M$. We set $\rho = M \times 2^\lambda$ for a security parameter λ so that a random $x \in \mathbb{Z}_{2^\rho}$ yields an element $x \bmod M$ that is statistically close to random in R_{ptxt}.

Re-randomize. To re-randomize this encoding, add a random subset sum of the public encodings of zero available for the level. We choose the parameter τ, roughly, to be large enough so that the result is statistically close to a fresh random encoding. To determine τ, several things need to be taken into account:

- CLT13 multilinear maps are noisy. Therefore, τ needs to be large enough so that, after adding a random subset sum, the noise terms *of all encodings in the meta-encoding* are statistically close to random, and independent of the original noise. A lower bound τ_0 on τ can be computed using the "leftover hash lemma over lattices" [AGHS13]. This requires the noise of the post-re-randomization encodings to be noticeably larger than the original noise. Therefore, there are actually two noise values for our scheme, one for the public parameters and one for the encodings produced by re-randomization, which are the actual encodings users will produce.
- The enforcing matrices need to be statistically close to random fresh enforcing matrices. A lower bound τ_1 on τ for this use can be computed using the standard leftover hash lemma.

By setting $\tau = \tau_0 + \tau_1$, we can guarantee that a re-randomized encoding is statistically close to a new fresh encoding of the same element.

Zero-test and Extract. Zero testing and extraction on top-level encodings (which are just top-level CLT13 encodings) are performed exactly as in CLT13.

5.2 Enforcing Matrix Structure

We now describe the enforcing matrix structure used in the matrices of our scheme. Consider the ℓ matrices associated to an encoding at any singleton level $\{u\}$ for $u \in [d]$, which have a block diagonal form. For each matrix, all the diagonal entries except the top left three entries are responsible for providing the enforcing structure. This enforcing block is itself a block diagonal matrix, divided into d matrices. $d-1$ of these blocks set to random multiples of the identity matrix, and the uth block set to E_i. This E_i block provides the enforcing structure for level $\{u\}$, while the other blocks are set to the identity to avoid interfering with the enforcing structure of the other singleton levels.

To construct an E_i for a new encoding, we sample a random vector α of dimension ℓ. Denote the ith component as α_i. The matrix E_i is set to be the following diagonal matrix of width $2(\ell-1)$:

$$E_i = \mathrm{diag}(\alpha_i, \alpha_{\sigma_{(12)}(i)}, \alpha_i, \alpha_{\sigma_{(23)}(i)}, \ldots, \alpha_i, \alpha_{\sigma_{(\ell-1,\ell)}(i)})$$

Here, $\sigma_{(ab)}$ denotes the transposition swapping a and b. We additionally have two bookend vectors F, G, used by all enforcing matrices for a particular singleton level. The left bookend vector F is simply the all 1's row vector of dimension $2(\ell-1)$. The right bookend vector G is a column vector of dimension $2(\ell-1)$. We sample $\ell-1$ random values $\{\eta_i\}_{i\in[\ell-1]}$, and set the entry in position $2i-1$ to η_i, and the entry in position $2i$ to $-\eta_i$ for all $i \in [\ell-1]$.

Analysis We now consider a setting where the adversary has access to dk meta-encodings of various matrix plaintext vectors, k in each singleton level. These encodings form a $d\ell \times k$ grid, with k matrices in each of the $d\ell$ columns.

Applied to our construction, the matrices are the various meta-encodings of 0 and powers of 2 that the adversary is given in the public parameters. The adversary also has access to a pre-zero-test encoding, which does not fit this pattern. However, we can think of the pre-zero-test encoding as arising from d meta-encodings with matrix plaintext sequence $(0, 1, \ldots, 1)$, one encoding per singleton level. The actual pre-zero-test encoding is obtained by multiplying these encodings together.

Definition 5. *We define an integer combination product to be a product of matrices resulting from taking the same integer linear combination of matrices in each column, and multiplying all the resulting matrices*

together in column order. We restrict the allowed integer combinations to be the set of k dimensional vectors of non-negative integers that sum to exactly ℓ. Denote this set by $H_{\ell,k}$.

We let $C_i^{(u)}$ be the width $2d(\ell-1)$ diagonal matrix made up of d blocks of matrices. Essentially, these will be the matrices A_i defined earlier, except that the first three rows and columns will be deleted. Thus, $d-1$ of the blocks of $C_i^{(u)}$ are set to be width-$2(\ell-1)$ identity matrices (randomly scaled) and the uth block is set to E_i, where a fresh random set of E_i is generated for each $C^{(u)}$. Let $C^{(u)}$ denote the list of $C_i^{(u)}$ matrices. The bookends are formed by concatenating d independently generated instances of our $2(\ell-1)$ dimensional bookends.

Consider several different generations of these $C^{(u)}$, denoted $C_{(j)}^{(u)}$ as j ranges from 1 to k for some parameter k. Let $C_{i,j}^{(u)}$ denote the ith matrix in $C_{(j)}^{(u)}$. Notice that adding point-wise $C_{(j_0)}^{(u)}$ to $C_{(j_1)}^{(u)}$ or scaling $C_{(j)}^{(u)}$ preserves the form of the matrices (though now the α's are different). Notice also that multiplying all the matrices in $C_{(j)}^{(u)}$ together along with our bookend vectors, gives 0. Therefore, we can take arbitrary linear combinations of the $C_{(j)}^{(u)}$, multiply them together, and still get 0. Next, we show that this is essentially the only way to combine different $C_{i,j}^{(u)}$ to get zero.

Definition 6. *We define a valid monomial to be a polynomial representing a product of the $C_{i,j}^{(u)}$ matrices, so that exactly one matrix is taken from each column, in column order, along with the bookends.*

These enforcing matrices are designed so the following lemma holds:

Lemma 3. *Let Q be a linear combination of valid monomials. If Q evaluates to 0 as a polynomial over the underlying randomness of the $C_{i,j}^{(u)}$ matrices, then Q must in fact be a linear combination of integer combination products.*

Proof. First, we prove this theorem for the case where $d=1$, so we can suppress the superscript (u). Let $\alpha_{j,1},\ldots,\alpha_{j,\ell}$ be the sequence of α's used in $C_{(j)}^{(u)}$.

Consider an arbitrary monomial. We represent choosing a matrix from each column as the vector $v=(v_1,v_2,\ldots,v_\ell)$ where each $v_i\in[k]$. The matrix resulting from this product, multiplied by the bookend vectors is

$$\begin{bmatrix}1&1&1&1&\cdots\end{bmatrix}\begin{bmatrix}\prod_i^\ell \alpha_{v_i,i} & & & & \\ & \prod_i^\ell \alpha_{v_i,\sigma_{(12)}(i)} & & & \\ & & \prod_i^\ell \alpha_{v_i,i} & & \\ & & & \prod_i^\ell \alpha_{v_i,\sigma_{(23)}(i)} & \\ & & & & \ddots\end{bmatrix}\begin{bmatrix}\eta_1\\-\eta_1\\\eta_2\\-\eta_2\\\vdots\end{bmatrix}$$

We can multiply out the above matrix product, which allows us to write the general form of a polynomial that sums over all such v as

$$\sum_{v\in[k]^\ell}\beta_v\sum_{h=1}^{\ell-1}\eta_h\left(\prod_i^\ell\alpha_{v_i,i}-\prod_i^\ell\alpha_{v_i,\sigma_{(h,h+1)}(i)}\right),$$

We treat each η_h and $\alpha_{i,j}$ as distinct formal variables. If this polynomial equals zero, then each monomial over these variables must have a coefficient of zero. Consider the monomial $\eta_{h'}\prod_i^\ell\alpha_{v'_i,i}$ where h' is a fixed integer and v' is a fixed vector. Only two terms in the above sum contribute to this coefficient, giving $\beta_{v'}-\beta_{\sigma_{(h',h'+1)}(v')}=0$. The transpositions $(12),(23),\ldots,(\ell-1,\ell)$ generate the entire symmetric group S_ℓ. Thus, by considering all possible choices of h', we conclude that $\beta_v=\beta_{\sigma(v)}$ for any permutation $\sigma\in S_\ell$.

The constraint that β_v remains the same when v is permuted means that a general polynomial of the form $\sum_{v\in[k]^\ell} \beta_v \prod_{i\in[\ell]} C_{i,v_i}$ can be written as a linear combination of polynomials from the set

$$\left\{ \sum_{v=\sigma(\bar{v}),\sigma\in S_\ell} \prod_{i\in[\ell]} C_{i,v_i} \mid \bar{v}\in[k]^\ell, \bar{v} \text{ is non-decreasing} \right\}.$$

We let $I_{\ell,k}$ denote the set of non-decreasing vectors in $[k]^\ell$. Our general polynomial can be written using only $\beta_{\bar{v}}$ coefficients where $\bar{v}\in I_{\ell,k}$ as

$$\sum_{\bar{v}\in I_{\ell,k}} \beta_{\bar{v}} \sum_{v=\sigma(\bar{v}),\sigma\in S_\ell} \prod_{i\in[\ell]} C_{i,v_i} \ .$$

The last step is to show that this polynomial can in fact be written as a linear combination of integer combination products, where each integer combination vector has non-negative integer components that sum to ℓ. Let the set of such k-dimensional vectors x be denoted as $H_{\ell,k}$. The goal is to show that there exist μ_x coefficients such that

$$\sum_{\bar{v}\in I_{\ell,k}} \beta_{\bar{v}} \left(\sum_{v=\sigma(\bar{v}),\sigma\in S_\ell} \prod_{i\in[\ell]} C_{i,v_i} \right) = \sum_{x\in H_{\ell,k}} \mu_x \left(\prod_{i=1}^{\ell} \sum_{j=1}^{k} x_j C_{i,j} \right).$$

It turns out that we can prove a slightly stronger claim. Consider the space of all polynomials over the monomials $\left\{\prod_{i\in[\ell]} C_{i,v_i}\right\}_{v\in[k]^\ell}$ that are symmetric under permutations of v. It is clear that $\left\{\sum_{v=\sigma(\bar{v}),\sigma\in S_\ell} \prod_{i\in[\ell]} C_{i,v_i}\right\}_{\bar{v}\in I_{\ell,k}}$ give a set of basis vectors for this space of polynomials. We claim that $\left\{\prod_{i=1}^{\ell}\sum_{j=1}^{k} x_j C_{i,j}\right\}_{x\in H_{\ell,k}}$ are also a basis for this space. Observe that $|H_{\ell,k}| = |I_{\ell,k}| = \binom{\ell+k-1}{k-1}$.

Consider the $(\ell+k-1)\times(\ell+k-1)$ matrix $M_{\ell,k}$ where each row is indexed by a distinct element of $H_{\ell,k}$. Let each column also be indexed by a distinct element of $H_{\ell,k}$. To compute an element of this matrix, we consider the associated row index, a vector $x=(x_1,\ldots,x_k)$, and the associated column, a vector $y=(y_1,\ldots,y_k)$. The element is set to $\prod_{i=1}^{k} x_i^{y_i}$. Brunat and Montes show that this matrix has positive determinant over the integers for all positive integers ℓ, k where $k\geq\ell$ [BM07]. Moreover, they show all of the determinant's prime factors are at most ℓ, which is a polynomial. We work over the ciphertext ring $R_{\mathsf{ctxt}} = \mathbb{Z}_N$, where all the prime factors of N are exponentially large. Therefore, the determinant is a unit in R_{ctxt}. Hence, this matrix is invertible over R_{ctxt}.

The row corresponding to any $x\in H_{\ell,k}$ can be interpreted as the vector needed to write $\prod_{i=1}^{\ell}\sum_{j=1}^{k} x_j C_{i,j}$ as a linear combination of vectors from $\{\sum_{v=\sigma(\bar{v}),\sigma\in S_\ell}\prod_{i\in[\ell]} C_{i,v_i}\}_{x\in I_{\ell,k}}$. To see this, we consider the following bijection from the column indices, which are elements in $H_{\ell,k}$, to elements of $I_{\ell,k}$. Given an element $(y_1,y_2,\ldots,y_k)\in H_{\ell,k}$, we form a vector where the first y_1 entries are 1, the next y_2 entries are 2, etc., which gives an element in $I_{\ell,k}$. Given this bijection, the entry whose row corresponds to $x\in H_{\ell,k}$ and whose column corresponds to $\bar{v}\in I_{\ell,k}$ can be seen as $\prod_i^{\ell} x_{\bar{v}_i}$. This is precisely the coefficient of $\sum_{v=\sigma(\bar{v}),\sigma\in S_\ell}\prod_{i\in[\ell]} C_{i,v_i}$ when $\prod_{i=1}^{\ell}\sum_{j=1}^{k} x_j C_{i,j}$ is expanded out.

Thus, we have an invertible linear transformation from one set of vectors to the other, and therefore a general polynomial can be rewritten in the form $\sum_{x\in H_{\ell,k}} \mu_x \left(\prod_{i=1}^{\ell}\sum_{j=1}^{k} x_j C_{i,j}\right)$.

To complete the proof, we claim that the above argument for the one-input case generalizes easily to the d-input case. We return to the notation where $C_{i,j}^{(u)}$ is the jth matrix in the ith column associated to singleton set $\{u\}$. General polynomials are now of the form

$$\sum_{v^{(1)},\ldots,v^{(d)}\in[k]^\ell} \beta_{v^{(1)},\ldots,v^{(d)}} \prod_{i\in[\ell]} C_{i,v_i^{(1)}}^{(1)} \cdots \prod_{i\in[\ell]} C_{i,v_i^{(d)}}^{(d)} \ .$$

Note that the enforcing matrices are constructed so that all matrices *not* corresponding to set $\{i\}$ have (scaled) identity matrices in the set of positions used to enforce $\{i\}$. The analysis we perform in the $d=1$ case

to conclude that each β_v is constant under permutations of v therefore still applies. Thus, these polynomials can actually be written as

$$\sum_{\bar{v}^{(1)},\ldots,\bar{v}^{(n)} \in I_{\ell,k}} \beta_{\bar{v}^{(1)},\ldots,\bar{v}^{(n)}} \left(\sum_{v^{(1)}=\sigma^{(1)}(\bar{v}^{(1)}),\sigma^{(1)} \in S_\ell} \prod_{i \in [\ell]} C^{(1)}_{i,v_i^{(1)}} \right) \cdots \left(\sum_{v^{(d)}=\sigma^{(d)}(\bar{v}^{(d)}),\sigma^{(d)} \in S_\ell} \prod_{i \in [\ell]} C^{(n)}_{i,v_i^{(n)}} \right).$$

The goal is to write these polynomials as a linear combination of integer combination products, which requires finding $\mu_{x^{(1)},\ldots,x^{(n)}}$ variables so that the polynomial can be re-written as

$$\sum_{x^{(1)},\ldots,x^{(n)} \in H_{\ell,k}} \mu_{x^{(1)},\ldots,x^{(n)}} \left(\prod_{i=1}^{\ell} \sum_{j=1}^{k} x_j^{(1)} C^{(1)}_{i,j} \right) \cdots \left(\prod_{i=1}^{\ell} \sum_{j=1}^{k} x_j^{(n)} C^{(n)}_{i,j} \right).$$

The same linear transformation argument from the $n = 1$ case works. The set of polynomials

$$\left\{ \left(\sum_{v^{(1)}=\sigma^{(1)}(\bar{v}^{(1)}),\sigma^{(1)} \in S_\ell} \prod_{i \in [\ell]} C^{(1)}_{i,v_i^{(1)}} \right) \cdots \left(\sum_{v^{(d)}=\sigma^{(d)}(\bar{v}^{(d)}),\sigma^{(d)} \in S_\ell} \prod_{i \in [\ell]} C^{(d)}_{i,v_i^{(d)}} \right) \right\}_{\bar{v}^{(1)},\ldots,\bar{v}^{(d)} \in I_{\ell,k}}$$

is a basis of all polynomials where each monomial consists of one variable from each column, and that are symmetric with respect to permutation on each $v^{(u)}$.

The linear transformation matrix to the set of polynomials

$$\left\{ \left(\prod_{i=1}^{\ell} \sum_{j=1}^{k} x_j^{(1)} C^{(1)}_{i,j} \right) \cdots \left(\prod_{i=1}^{\ell} \sum_{j=1}^{k} x_j^{(d)} C^{(d)}_{i,j} \right) \right\}_{x^{(1)},x^{(d)} \in H_{\ell,k}}.$$

is simply the dth Kronecker product of $M_{\ell,k}$ with itself. Since $M_{\ell,k}$ is invertible in the ring R_{ctxt}, its dth Kronecker product is also invertible, finishing the proof.

5.3 Security

The security of this multilinear map will rest on the following assumption about annihilating vector-input branching programs.

Recall that we define $H_{\ell,k}$ to be the set of vectors of dimension k with non-negative integer components that sum to ℓ. Additionally, we refer to a generic vector-input branching program as one whose matrix elements are all distinct formal variables.

Assumption 1 (The (ℓ, s)-VBPUA Assumption) *Let $\ell = \mathsf{poly}(n, \lambda)$ and $s = \mathsf{poly}(n, \lambda)$ be parameters, and let $f(x)$ be a generic vector-input branching program that reads each input ℓ times and consists of $s \times s$ matrices. Let $\mathcal{A}$ denote a probabilistic exponential-time algorithm that, on input $(1^n, 1^\lambda)$, outputs a polynomial number of exponentially-sized polynomials of the form $P_r = \sum_{x \in H_{\ell,t}} \alpha_{r,x} f(x)$ where each set of vectors in $\{\alpha_r\}_r$ (interpreted as vectors over all x) are linearly independent. Then there does not exist any polynomial-sized circuit Q of degree at most $2^{o(\lambda)}$ such that $Q(\{P_r\}_r) \equiv 0$ as a polynomial over the formal variables of the generic VBP.*

To prove the security of this multilinear map, we consider two worlds. Suppose the multilinear map is used to encode a sequence $m_1, \ldots, m_v$ of plaintexts at levels $S_1, \ldots, S_v$.

In the "real" world, the adversary interacts with our weak CLT13 model, whose plaintexts consist of all the elements of all the matrices output by our multilinear map encoding procedure as well as all the elements of the public parameter matrices. These plaintexts are encoded at the appropriate levels derived

from $S_1, \ldots, S_v$. The adversary can submit level-respecting polynomials over these plaintexts as zero-test queries, and receives a handle to the result of the zero-test computation for successful queries. Then the adversary enters a post-zero test stage, and can automatically win by submitting a polynomial Q that satisfies the necessary conditions.

In the "ideal" world, the adversary interacts with a simulator that can interact with a vanilla generic multilinear map model. In this world, the model only stores the actual plaintexts $m_1, \ldots, m_v$ and levels $S_1, \ldots, S_v$. The adversary submits level-respecting polynomials over the plaintexts of the real world. The simulator answers these queries using only queries to its model over the actual plaintexts. As in the real world, the adversary enters a post-zero test stage, which the simulator must respond to.

The multilinear map is secure as long as (1) the adversary can never create a successful polynomial Q in the real world, and (2) any information the adversary gets in the real world, the adversary can also obtain in the ideal world. Formally, this requires proving the existence of a simulator such that no PPT adversary can distinguish between the two worlds.

Theorem 2. *There exists a simulator S such that for all PPT adversaries $\mathcal{A}$,*

$$\Pr[\texttt{EXP}_{real}(\mathcal{A}) = \texttt{EXP}_{ideal}(S, \mathcal{A})] = 1 - negl$$

Moreover, S always responds to post-zero test queries with 0.

Proof. First, we rule out the case where there are no plaintexts to be encoded. In this case, the vanilla generic multilinear map model is vacuous, and the only way to win is for the adversary to come up with a polynomial Q causing the weak CLT13 model to declare a win. We invoke Theorem 1, which states that if $\mathcal{A}$ can win, then there exists another adversary $\mathcal{A}'$ that wins in the CLT13 annihilating model. We will show that the existence of such an adversary $\mathcal{A}'$ violates the (ℓ, s)-VBPUA assumption. As in the previous section, for simplicity, we will imagine the pre-zero-test encoding given out as a set of d meta-encodings at the singleton levels. Forcing them to be pre-multiplied only makes the adversary's task harder.

Consider an annihilating polynomial Q' that $\mathcal{A}'$ submits in order to win in the plain annihilating model. Since $\mathcal{A}'$ must construct this polynomial from the results of successful zero-tests on elements from the public parameters, the level structure imposes some restrictions on the allowed monomials. Recall that the matrices in the public parameters are encoded so that all elements of all matrices in column i are encoded at level $\{i\}$ (and the two bookends are encoded at level $\{0\}$ and $\{n\ell + 1\}$ respectively), and that the top level of the map is $\{0, 1, \ldots, n\ell + 1\}$. This means each monomial is a product of $n\ell + 2$ matrix entries, such that exactly one entry is chosen from each of the $n\ell$ columns, plus two entries chosen from the bookend vectors. Note that there may be exponentially many monomials in this polynomial.

Our next step is to apply Lemma 2 (stated in Section 3 and proved in Appendix A). The collection of matrices $\{A_{i,j}\}$ in the lemma statement are set to be the matrices given out in the public parameters. The set S of allowable products will be the exponentially-sized set that simply allows all level-respecting products (i.e. one element/matrix from each column). Additionally non-shortcutting is satisfied: the v_i, w_i entries in the second and third positions in each matrix are all distinct. This means, when treated as formal variables, they cause any iterated matrix product not including one of the bookends to contain a unique product of these values. Hence, all partial iterated matrix products are linearly independent. Applying the lemma to the annihilating polynomial Q' tells us that Q' is in fact a linear combination of matrix products, where each product consists of exactly one matrix chosen from each column (and the bookends).

The lemma also tells us that Q', evaluated over the original un-randomized matrices, is zero. Now we invoke the enforcing matrices. Q' satisfies the conditions required to invoke Lemma 3, from which we conclude that Q' is a linear combination of integer combination products. Furthermore, the types of integer combinations that are allowed are precisely those that live on the simplex.

Observe that we can view the public parameters of multilinear map as the matrices in a vector-input branching program (VBP). Each integer combination product of the multilinear map matrices is just the evaluation of this VBP on a simplex input vector. The (ℓ, s)-VBPUA assumption states that there does not exist a polynomial that can annihilate a linear combination of VBP evaluations on simplex input vectors, which directly contradicts the existence of Q'.

We conclude that $\mathcal{A}'$ cannot win in the annihilating model with non-negligible probability, and thus $\mathcal{A}$ cannot win in the weak model.

Now we extend to the case where $\mathcal{A}$ and $\mathcal{A}'$ obtain some encodings outputted by the encoding procedure. However, recall that the encoding procedure just computes a linear combination over the terms in the public parameters. Therefore, if $\mathcal{A}$ or $\mathcal{A}'$ can win (i.e. produce a successful post-zero test polynomial Q or Q', respectively), then they can be used to construct adversaries $\mathcal{B}$ and $\mathcal{B}'$ that only require the public parameters.

At this point, we have shown that no adversary in our weak CLT13 model can create a successful post-zero-test query. Therefore, we can effectively ignore these queries, as the simulator can always respond with "non-zero".

Thus, for $\mathcal{A}$ to distinguish between the two real and ideal worlds, it must do so entirely based on information from the zero-test queries. In other words, it must attack our scheme in the vanilla generic graded encoding model. It now suffices to give a simulator S that can answer $\mathcal{A}$'s queries in the ideal world the same way that they would be answered in the real world.

The simulator S works as follows. $\mathcal{A}$ submits a polynomial Q to S for zero-testing. This polynomial Q is over the public parameters of the multilinear map, as well as the encodings of the plaintexts, where we denote the ith plaintext for level $\{u\}$ as $m_i^{(u)}$. First, the simulator S attempts to re-express the polynomial in the form $\sum_{x \in H_{\ell,k}} \mu_x P_x$, where each P_x is the result of taking the integer linear combination of matrices specified by x in each column, and multiplying them all together. Note that this step of S may be computationally inefficient, but we will allow this in the vanilla generic multilinear map model.

If the polynomial cannot be re-expressed in this form, our previous analysis implies that it is not zero with high probability. At this point, the simulator S can simply return "not zero". Otherwise, we know that in order for the query to be zero (except with negligible probability), it must be that the polynomial, when evaluated on the pre-Kilian-randomized A matrices, gives identically 0. Suppose the kth matrix corresponding to meta-level u encodes the sequence of matrix plaintext $(m_{i,1}^{(u)}, m_{i,2}^{(u)}, \ldots, m_{i,\ell}^{(u)})$. Then, using the structure of our matrices, the polynomial, when evaluated on the A matrices, outputs

$$\sum_{x \in H_{\ell,k}} \mu_x \left(\prod_{u=1}^{d} \prod_{j=1}^{\ell} (\sum_i x_i m_{i,j}^{(u)}) \right) .$$

For the matrices belonging to the public parameters, the $m_{i,j}^{(u)}$ are just known constants derived from the structure of the public parameters. For the matrices belonging to singleton encodings, the simulator knows that $m_{i,j}^{(u)} = 0$ for $j > 1$, and that $m_{i,1}^{(u)}$ corresponds to the unknown plaintext m_v encoded by the meta-encoding. Thus, by hardcoding the appropriate constants and re-naming variables, S can construct a polynomial X over the m_v terms which has total degree at most d. Notice that since we are restricting to a constant-degree multilinear map, this polynomial can be represented as a linear combination of its monomials, and this representation only has polynomial size. Thus, this query can be implemented as a polynomial-sized circuit. Therefore, S can submit X to its multilinear map oracle to determine whether X evaluates to zero on the m_v. With overwhelming probability, the answer is the same as whether Q evaluates to zero. Thus, S successfully answers each of the adversary's queries.

5.4 Extending Our Maps to Allow Intermediate Levels

We now briefly describe how to extend our scheme to allow intermediate levels. Such intermediate levels allow for a richer level structure that is needed by many applications.

First, notice that our construction naturally allows for level $\{u, u+1\}$ meta-encodings for any u. Such encodings are obtained from a level u and level $u+1$ meta-encoding by multiplying the ith matrix of level u by the ith matrix of level $u+1$, for each i. The structure of such intermediate encodings can be inferred from the structure of the singleton meta-encodings. Essentially, they have the same structure as singleton encodings, except now two of the block diagonal terms are set to be (independent) enforcing matrices.

These higher level encodings can easily be generalized to any interval $[a, b] \subset [d]$. Unfortunately, due to having different Kilian randomizing matrices for each column, we cannot directly multiply level u and $u+2$ meta-encodings to obtain a level $\{u, u+2\}$ encoding in this fashion. Instead, we observe that instead of multiplying the underlying matrices, we can *tensor* the matrices, obtaining a list of all degree-2 products. The structure here is slightly more complicated, but can still be inferred from the structure of singleton meta-encodings. Now a tensored level-$\{u, u+2\}$ encoding can be multiplied, say, by a level $u+1$ encoding, arriving at a level $[u, u+2]$. This is done be using the degree-2 products to re-construct the necessary 3-way matrix product.

This tensoring can be extended to any subset of $[d]$, though the size of the encodings grows exponentially in d. However, since we keep d a constant, the size of encodings never grows beyond a polynomial.

Finally, we can allow re-randomization at any encoding level by giving out many encodings of zero at that level. The number of encodings will need to be much higher in order to completely re-randomize, as now there are many more terms that need to be random.

While tedious, we can also extend our security proof from the preceding section to handle this new setting. The idea is similar to how we handled the pre-zero-test encoding. Basically, we can express the encodings in the public parameters as the result of polynomial computations on many bottom-level terms. This allows us to apply the proof from the previous section. We omit the precise details.

6 Acknowledgements

We thank Leon Zhang (UC Berkeley Math) for helpful discussions regarding the proof of Lemma 3.

References

[ABD16] Martin R. Albrecht, Shi Bai, and Léo Ducas. A subfield lattice attack on overstretched NTRU assumptions - cryptanalysis of some FHE and graded encoding schemes. pages 153–178, 2016.

[AFH+16] Martin R. Albrecht, Pooya Farshim, Dennis Hofheinz, Enrique Larraia, and Kenneth G. Paterson. Multilinear maps from obfuscation. pages 446–473, 2016.

[AGHS13] Shweta Agrawal, Craig Gentry, Shai Halevi, and Amit Sahai. Discrete Gaussian leftover hash lemma over infinite domains. pages 97–116, 2013.

[AS16] Prabhanjan Vijendra Ananth and Amit Sahai. Functional encryption for turing machines. pages 125–153, 2016.

[BGH+15] Zvika Brakerski, Craig Gentry, Shai Halevi, Tancrède Lepoint, Amit Sahai, and Mehdi Tibouchi. Cryptanalysis of the quadratic zero-testing of GGH. Cryptology ePrint Archive, Report 2015/845, 2015. http://eprint.iacr.org/2015/845.

[BGK+14] Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Protecting obfuscation against algebraic attacks. pages 221–238, 2014.

[BLR+15] Dan Boneh, Kevin Lewi, Mariana Raykova, Amit Sahai, Mark Zhandry, and Joe Zimmerman. Semantically secure order-revealing encryption: Multi-input functional encryption without obfuscation. pages 563–594, 2015.

[BM07] Josep M Brunat and Antonio Montes. A polynomial generalization of the power-compositions determinant. *Linear and Multilinear Algebra*, 55(4):303–313, 2007.

[BMSZ16] Saikrishna Badrinarayanan, Eric Miles, Amit Sahai, and Mark Zhandry. Post-zeroizing obfuscation: New mathematical tools, and the case of evasive circuits. pages 764–791, 2016.

[BPR15] Nir Bitansky, Omer Paneth, and Alon Rosen. On the cryptographic hardness of finding a Nash equilibrium. pages 1480–1498, 2015.

[BPW16] Nir Bitansky, Omer Paneth, and Daniel Wichs. Perfect structure on the edge of chaos - trapdoor permutations from indistinguishability obfuscation. pages 474–502, 2016.

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

[BS03] Dan Boneh and Alice Silverberg. Applications of multilinear forms to cryptography. *Contemporary Mathematics*, 324:71–90, 2003.

[BWZ14a] Dan Boneh, Brent Waters, and Mark Zhandry. Low overhead broadcast encryption from multilinear maps. pages 206–223, 2014.

[BWZ14b] Dan Boneh, David J. Wu, and Joe Zimmerman. Immunizing multilinear maps against zeroizing attacks. Cryptology ePrint Archive, Report 2014/930, 2014. http://eprint.iacr.org/2014/930.

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

[BZ16] Mark Bun and Mark Zhandry. Order-revealing encryption and the hardness of private learning. pages 176–206, 2016.

[CDPR16] Ronald Cramer, Léo Ducas, Chris Peikert, and Oded Regev. Recovering short generators of principal ideals in cyclotomic rings. pages 559–585, 2016.

[CFL+16] Jung Hee Cheon, Pierre-Alain Fouque, Changmin Lee, Brice Minaud, and Hansol Ryu. Cryptanalysis of the new CLT multilinear map over the integers. pages 509–536, 2016.

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

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

[CJL16] Jung Hee Cheon, Jinhyuck Jeong, and Changmin Lee. An algorithm for NTRU problems and cryptanalysis of the GGH multilinear map without a low level encoding of zero. Cryptology ePrint Archive, Report 2016/139, 2016. http://eprint.iacr.org/2016/139.

[CLLT16a] Jean-Sébastien Coron, Moon Sung Lee, Tancrède Lepoint, and Mehdi Tibouchi. Cryptanalysis of GGH15 multilinear maps. pages 607–628, 2016.

[CLLT16b] Jean-Sébastien Coron, Moon Sung Lee, Tancrède Lepoint, and Mehdi Tibouchi. Zeroizing attacks on indistinguishability obfuscation over CLT13. Cryptology ePrint Archive, Report 2016/1011, 2016. http://eprint.iacr.org/2016/1011.

[CLLT17] Jean-Sébastien Coron, Moon Sung Lee, Tancrède Lepoint, and Mehdi Tibouchi. Zeroizing attacks on indistinguishability obfuscation over CLT13. pages 41–58, 2017.

[CLP15] Kai-Min Chung, Huijia Lin, and Rafael Pass. Constant-round concurrent zero-knowledge from indistinguishability obfuscation. pages 287–307, 2015.

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

[CLT15] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. New multilinear maps over the integers. pages 267–286, 2015.

[GGH13a] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. 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. pages 40–49, 2013.

[GGH+13c] Sanjam Garg, Craig Gentry, Shai Halevi, Amit Sahai, and Brent Waters. Attribute-based encryption for circuits from multilinear maps. pages 479–499, 2013.

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

[GGHZ16] Sanjam Garg, Craig Gentry, Shai Halevi, and Mark Zhandry. Functional encryption without obfuscation. pages 480–511, 2016.

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

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

[Hal15] Shai Halevi. Graded encoding, variations on a scheme. Cryptology ePrint Archive, Report 2015/866, 2015. http://eprint.iacr.org/2015/866.

[HJ16] Yupu Hu and Huiwen Jia. Cryptanalysis of GGH map. pages 537–565, 2016.

[HSW14] Susan Hohenberger, Amit Sahai, and Brent Waters. Replacing a random oracle: Full domain hash from indistinguishability obfuscation. pages 201–220, 2014.

[HW15] Pavel Hubacek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. pages 163–172, 2015.

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

[LMA+16] Kevin Lewi, Alex J. Malozemoff, Daniel Apon, Brent Carmer, Adam Foltzer, Daniel Wagner, David W. Archer, Dan Boneh, Jonathan Katz, and Mariana Raykova. 5Gen: A framework for prototyping applications using multilinear maps and matrix branching programs. pages 981–992, 2016.

[MSZ16] Eric Miles, Amit Sahai, and Mark Zhandry. Annihilation attacks for multilinear maps: Cryptanalysis of indistinguishability obfuscation over GGH13. pages 629–658, 2016.

[PPS15] Omkant Pandey, Manoj Prabhakaran, and Amit Sahai. Obfuscation-based non-black-box simulation and four message concurrent zero knowledge for NP. pages 638–667, 2015.

[PS15] Omer Paneth and Amit Sahai. On the equivalence of obfuscation and multilinear maps. Cryptology ePrint Archive, Report 2015/791, 2015. http://eprint.iacr.org/2015/791.

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

A Proof of Lemma 2

Proof. We prove this by induction on the number of columns. The base case is where the number of columns is one. There are no R_i matrices and every matrix must be a 1×1 matrix, and thus any allowable polynomial is immediately an allowable matrix product.

For the inductive step, we assume that this lemma holds for k columns and any set S. We let s denote an element of S, and we let $s(i)$ denote the ith element of tuple s. For notational convenience, we use $A_{s(i)}$ as shorthand for the matrix $A_{i,s(i)}$. $A_{s(i),i',j'}$ will denote the (i',j') entry of matrix $A_{s(i)}$.

Consider an arbitrary allowable polynomial:

$$p = \sum_{s\in S, i_1,i_2,j_2,\ldots,i_n} \alpha_{s,i_1,i_2,j_2,\ldots,i_n} A_{s(1),1,j_1} \left(\prod_{k\in[n-1]\setminus\{1\}} A_{s(k),i_k,j_k} \right) A_{s(n),i_n,1}$$

We can expand out this polynomial in terms of the matrix R_1:

$$p = \sum_{s\in S, i_1,i_2,j_2,\ldots,i_n,m,\ell} \alpha_{s,i_1,i_2,j_2,\ldots,i_n} A_{s(1),1,m} R^{adj}_{1,m,j_1} R_{1,i_2,\ell} (\mathbf{A}_{s(2)} \cdot \mathbf{R}^{adj}_2)_{\ell,j_2} \left(\prod_{k\in[n-1]\setminus\{1,2\}} A_{s(k),i_k,j_k} \right) A_{s(n),i_n,1}$$

We define a new coefficient:

$$\alpha'_{s,j_1,i_2,\ell} = \sum_{j_2,\ldots,i_n} \alpha_{s,j_1,i_2,j_2,\ldots,i_n} (\mathbf{A}_{s(2)} \cdot \mathbf{R}^{adj}_2)_{\ell,j_2} \left(\prod_{k\in[n-1]\setminus\{1,2\}} A_{s(k),i_k,j_k} \right) A_{s(n),i_n,1}$$

This allows us to rewrite our polynomial as

$$p = \sum_{s\in S, j_1,i_2,m,\ell} \alpha'_{s,j_1,i_2,\ell} A_{s(1),1,m} R^{adj}_{1,m,j_1} R_{1,i_2,\ell}$$

We can expand p by substituting the following expression for the entries of the adjoint matrices:

$$R^{adj}_{1,m,j_1} = \sum_{\sigma:\sigma(m)=j_1} \mathsf{sign}(\sigma) \left(\prod_{t\neq m} R_{1,\sigma(t),t} \right).$$

This substitution gives

$$p = \sum_{s\in S, i_2,m,\ell,\sigma} \mathsf{sign}(\sigma) \alpha'_{s,\sigma(m),i_2,\ell} A_{s(1),1,m} \left(\prod_{t\neq m} R_{1,\sigma(t),t} \right) R_{1,i_2,\ell}$$

Since p is identically zero, the coefficients of any product of $\left(\prod_{t\neq m} R_{1,\sigma(t),t}\right) R_{1,i_2,\ell}$ terms must be 0. We consider the possible types of products we can have.

– Well-formed products. These products arise when the unselected entry $R_{1,\sigma(m),m}$ is reselected as $R_{1,i_2,\ell}$. The same product results when the permutation is fixed and the un-selected entry varies. This forces $\ell = m$ and $i_2 = \sigma(m)$. For each σ, the constraint that the coefficient of this product is zero gives the following expression:

$$\sum_{s\in S,m} \alpha'_{s,\sigma(m),\sigma(m),m} A_{s(1),1,m} = 0$$

Since we can pick σ so that $\sigma(m)$ ranges over all possible values of i, we conclude that for all i,

$$\sum_{s\in S,m} \alpha'_{s,i,i,m} A_{s(1),1,m} = 0$$

– Malformed Type 1. In these products, we unselect an entry and reselect one in a different column and row. This fixes $\ell \neq m$ and $i_2 \neq \sigma(m)$. Write $\sigma(m) = j_1$. Then for any choice of $\ell \neq m, i_2 \neq j$, we have

$$\sum_{s\in S} \alpha'_{s,j_1,i_2,\ell} A_{s(1),1,m} = 0$$

– Malformed Type 2. (Note that these are modified from the original malformed type 2 products defined in [BMSZ16]; here we select a different entry in the same row instead of column)
In these products, we unselect the entry and then select a different entry in the same row. Consider some other choice of i'_2, m', ℓ', σ' that leads to the same product. We know that $\ell' = \ell$ is uniquely determined as the column with two entries. Furthermore, we know that $m = m'$ is the only column with no entries. It remains to consider the $i_2 \neq i'_2$. case. Then σ' and σ have opposite parity. So we choose any σ that satisfies the following $i_2 \neq i'_2$, $\ell = \ell'$, $\sigma(m) = i_2$, $\sigma'(m') = i'_2$, and $\ell \neq m$.
So we have that for any fixed choice of $i_2, i'_2 \neq i_2, m \neq \ell$, that

$$\sum_{s\in S} (\alpha'_{s,i_2,i_2,\ell} - \alpha'_{s,i'_2,i'_2,\ell}) A_{s(1),1,m} = 0$$

– Malformed Type 3. (Again, note these are modified from the original malformed type 3 products defined in [BMSZ16]). In these products, we select something in an entirely different row. In other words, these are products of the form $\left(\prod_{t\neq m} R_{1,\sigma(t),t}\right) R_{1,i_2,\ell}$ where $i_2 \neq \sigma(m)$. The equations that result are redundant, so we do not consider them.

We can expand the equations resulting from the malformed type 1 products with the definitions. This gives the constraint that for any $j_1 \neq i_2, \ell \neq m$,

$$\sum_{s\in S, j_2,\ldots,i_n} \alpha_{s,j_1,i_2,j_2,\ldots,i_n} A_{s(1),1,m} (\mathbf{A}_{s(2)} \cdot \mathbf{R}_2^{adj})_{\ell,j_2} \left(\prod_{k\in[n-1]\setminus\{1,2\}} A_{s(k),i_k,j_k} \right) A_{s(n),i_n,1} = 0$$

Fix specific values of m, j_1, i_2. Now we apply the inductive hypothesis to the case where

$$\mathbf{A}'_{1,j} = (\mathbf{A}_{2,j} \cdot \mathbf{R}_2^{adj})_\ell \text{ for } 1 \leq j \leq a(2)$$
$$\mathbf{A}'_{\gamma,j} = \mathbf{A}_{\gamma+1,j} \text{ for } 2 \leq \gamma \leq n-1, 1 \leq j \leq a(\gamma)$$

The set S' of allowable products is simply the set of all $n-1$-tuple suffixes of n-tuples in S. Note that if we assume the non-shortcutting properties of the original matrices, the matrices resulting from this transformation satisfy non-shortcutting as well.

To apply this induction, we absorb the $A_{s(1),1,m}$ term into the α_s coefficients. Thus, when we apply induction, the coefficients are:

$$\beta_{s',j_2,i_3,j_3,\ldots,i_n} = \sum_{s \in S | s' \text{ is a suffix of } s} \alpha_{s,j_1,i_2,j_2,\ldots,i_n} A_{s(1),1,m} \text{ for } s' \in S'$$

The inductive hypothesis states that this polynomial must be a valid matrix product polynomial. So in particular, it states that any $\beta_{s',j_2,i_3,j_3,\ldots,i_n}$ for any $s' \in S'$ that satisfies $j_2 = i_3, j_3 = i_4, \ldots, j_{n-1} = i_n$ can in fact be written as $\beta_{s'}$, a term that has no dependence on $j_2, \ldots, i_n$. Furthermore, if any of $j_2 = i_3$ or $j_3 = i_4$, etc. fail to hold, then the coefficient is 0.

Now we label each $s' \in S'$ with a distinct number from $1, \ldots, |S'|$. Define the matrix M with entries

$$M_{\ell,q} = (A_{s'(1)})_\ell \left(\prod_{k \in [n-1]} A_{s'(k)} \right) \text{ for } s' \in S' \text{ with label } q$$

The right non-shortcutting property implies the columns of this matrix are linearly independent. Thus, this matrix has rank $|S'|$. Define the column vector where the qth entry is $\beta_{s'}$ for s' with label q. Observe the inner product of each row of M with this vector is 0, as the product is precisely the quantity in the malformed type 1 equations. Since there are $|S'|$ linearly independent rows, this vector must be the 0 vector.

The fact that $\beta_{s'} = 0$ for all $s' \in S'$ tells us that

$$\sum_{s' \in S'} \beta_{s'} = \sum_{s \in S} \alpha_{s,j_1,i_2,j_2,\ldots,i_n} A_{s(1),1,m} = 0$$

for fixed j_1, i_2, and $j_2, \ldots, i_n$ that satisfy $j_2 = i_3, j_3 = i_4, \ldots, j_{n-1} = i_n$.

Now we label each $s \in S$ with a distinct number from $1, \ldots, |S|$. Define the matrix W with entries

$$W_{m,r} = A_{s(1),1,m} \text{ for } s \in S \text{ with label } r$$

The left non-shortcutting property implies that all the columns are linearly independent, and thus this matrix has rank $|S|$. Define the column vector where the rth entry is $\alpha_{s,j_1,i_2,j_2,i_3,j_3,\ldots,i_n}$ (for the same fixed for fixed $j_1, i_2, j_2, \ldots, i_n$) for s with label r. Then the inner product of this vector with any row of W is 0. Since there are $|S|$ linearly independent rows, this vector must be the 0 vector.

So we conclude that each coefficient $\alpha_{s,j_1,i_2,j_2,\ldots,i_n}$ is zero if $j_1 \neq i_2$ for any superscript S.

We can apply this exact reasoning to the equations generated by the malformed type 2 products, and conclude that for any choice of $s \in S, j_2, \ldots, i_n$, and $i_2 \neq i_2'$,

$$\alpha_{s,i_2,i_2,j_2,\ldots,i_n} = \alpha_{s,i_2',i_2',j_2,\ldots,i_n}.$$

If we consider any general polynomial p over all possible monomials, the coefficient of any monomial that would not result from a proper multiplication of two matrices in the first two columns is zeroed out. Furthermore, if the coefficients "match up" so that all the remaining monomials are the result of a valid multiplication of two matrices in the first two columns.

Thus, any polynomial p is in fact a general polynomial over the following collection of matrices with $n-1$ columns. The first column is the set of all pairwise products of matrices in the original first two columns such that the product appears as the first two indices for some $s \in S$. The remaining $n-2$ original columns are left the same. The set of allowable products has the same size as the original S, but with each n-tuple rewritten as an $n-1$-tuple where the first index of the tuple corresponds to the matrix that results from multiplying the first two indices of the original tuple.

The inductive hypothesis is that any general polynomial over this collection of matrices is a valid matrix product polynomial. Note that valid matrix product polynomials over this collection of matrices are also valid matrix product polynomials over the original collection of matrices.

Finally, we note that the equations resulting from the well-formed product immediately imply that this polynomial evaluates to 0. This completes the induction.