The MMap Strikes Back: Obfuscation and New Multilinear Maps Immune to CLT13 Zeroizing Attacks

Fermi Ma* and Mark Zhandry**

Princeton University

Abstract. All known multilinear map candidates have suffered from a class of attacks known as "zeroizing" attacks, which render them unusable for many applications. We provide a new construction of *polynomial-degree* multilinear maps and show that our scheme is provably immune to zeroizing attacks under a strengthening of the Branching Program Un-Annihilatability Assumption (Garg et al., TCC 2016-B).

Concretely, we build our scheme on top of the CLT13 multilinear maps (Coron et al., CRYPTO 2013). In order to justify the security of our new scheme, we devise a weak multilinear map model for CLT13 that captures zeroizing attacks and generalizations, reflecting *all known* classical polynomial-time attacks on CLT13. In our model, we show that our new multilinear map scheme achieves *ideal security*, meaning no known attacks apply to our scheme. Using our scheme, we give a new multiparty key agreement protocol that is several orders of magnitude more efficient that what was previously possible.

We also demonstrate the general applicability of our model by showing that several existing obfuscation and order-revealing encryption schemes, when instantiated with the CLT13 maps, are secure against known attacks. These are schemes that are actually being implemented for experimentation, but until our work had no rigorous justification for security.

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 [1], attribute-based encryption for circuits [2], asymptotically optimal broadcast encryption [3], witness encryption [4], functional encryption [5, 6], and most notably mathematical program obfuscation [5]. In turn, obfuscation has been used to construct many more amazing applications [7–14] as well as establish interesting connections to other areas of computer science [15, 16].

Unfortunately, all known multilinear maps for degree $d > 2$ [17–19] have suffered from devastating attacks known as "zeroizing" attacks [17, 20–22]. These attacks have rendered most of the applications above insecure. In response, many authors introduced "fixes"; these fixes came in many forms, from tweaking how information is extracted from the map [23] to compiling the existing weak multilinear maps into new ones that were presumably stronger [6, 24, 25]. However, these fixes were largely ad hoc, and indeed it was quickly shown how to generalize the zeroizing attacks to circumvent the fixes [24, 26, 27, 25, 28].

Given the 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. [29] is that all (classical polynomial-time[1]) attacks require the ability to obtain an encoding of zero. Miles, Sahai, and Zhandry [33] observed moreover that all attacks on the original GGH13 multilinear map have a very similar structure. They define an abstract attack model, called the "annihilating attack model," that encompasses and generalizes all existing attacks on these specific maps. Remarkably, since their initial publication, all subsequent attacks found on GGH13 have fit within the annihilating model (also known as

* fermima1@gmail.com

** mzhandry@princeton.edu

[1] Sub-exponential and quantum attacks have been discovered on multilinear map candidates [30–32]. 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.

the "weak model"). Therefore, this annihilating model appears to be a fully general abstraction of known attack techniques for the GGH13 multilinear maps.

Within the weak model, Badrinaryanan et al. [29] constructed a secure witness encryption scheme, while Garg et al. [34] built secure obfuscation and order-revealing encryption. Since these constructions have been proven secure in the weak model, they are 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.

Give new applications of multilinear maps that can be constructed and proven 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 [35], and therefore most likely to eventually become usable in practice.

In addition, the vulnerabilities of existing multilinear map candidates lead to a natural third goal:

Construct multilinear maps that are not vulnerable to zeroizing attacks.

There has been some initial progress toward this goal. Several authors [36, 37] have shown how to construct a version of multilinear maps from obfuscation, which can in turn be built from weak multilinear maps using the aforementioned constructions. 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 by using weak maps to build strong maps. We do not use obfuscation, though our scheme is inspired by 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:

Weak CLT13 model. 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 from the model for the GGH13 maps, owing to the different technical details of the attacks. Unlike the GGH13 case, where the common thread amongst all the attacks was rather explicit,[3] the common features of CLT13 attacks are a bit more nebulous, and require additional effort to pull out and formalize.

Model Conversion Theorem. 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. This "plain annihilating model" makes it very easy to test if a particular usage of CLT13 is safe. For example, it is immediate from prior results that an *existing* class of obfuscation constructions [38, 29] is secure in the plain annihilating model, under the

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

[3] Namely, all attacks compute the ideal $\langle g \rangle$ generated by some element g.

same algebraic complexity assumption as in [34]. Hence, these schemes are also secure in our weak CLT13 model. This is the first rigorous argument for security of these schemes. We note that these obfuscation constructions are currently being implemented [35], so justifying their security is important.

Note that [38, 29] are *not* secure in the weak GGH13 model, indicating that the weak CLT13 model may be somewhat more useful.

New Multilinear Map Scheme. Armed with a weak model for CLT13, we devise a new *polynomial-degree* multilinear map scheme built on top of CLT13. [4] We then prove that within our weak CLT13 model, *there are no attacks on our new scheme* under a new Vector-Input Branching Program Un-Annihilatability (VBPUA) Assumption. That is, under our new assumption, 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 significantly more efficient than obfuscation-based multilinear maps, at least for simple settings.

Concretely, to implement a 4-Party Non-Interactive Key Exchange with 80 bits of security, our (fully optimized) scheme requires approximately 2^{14} CLT13 encodings with degree 2000. For comparison, the most efficient obfuscation-based approach requires at least 2^{44} CLT13 encodings from a map of twice the degree [39]. See Table 1 for more details. While still somewhat impractical, our scheme is a significant improvement on what is possible with current obfuscation constructions.

Scheme	MMap Degree	Public Encodings
Boneh et al. [39]	4150	2^{44}
Our map (setting 1)	52	2^{62}
Our map (setting 2)	160	2^{33}
Our map (setting 3)	1040	2^{19}
Our map (setting 4)	2000	2^{14}

Table 1. Approximate Parameter Sizes for 4-Party Non-Interactive Key Exchange (with $\lambda = 80$). We compare the parameter sizes of the obfuscation scheme due to Boneh et al. [39] to our construction. Note that Boneh et al. also need to encrypt a large program using fully homomorphic encryption, which we do not need; we do not include this in the comparison. Setting 1 is our plain construction with no optimizations. Setting 2 is a modified version of our construction that uses a common technique to re-randomize with fewer encodings. Settings 3 and 4 are further optimized to take advantage of CLT13 "slots." "MMap Degree" refers to the degree of the underlying multilinear map required to instantiate each scheme. "Public Encodings" is the number of public multilinear map encodings needed for the key exchange. These optimizations and estimates are detailed in Section 7.

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.

The notable limitation of our multilinear map construction is that the security analysis relies on a new algebraic complexity assumption about annihilating polynomials, which we call the Vector-Input Branching Program Un-Annihilatability (VBPUA) Assumption. The assumption is similar to the Branching Program Un-Annihilatability (BPUA) Assumption used in [34] (which is implied by the existence of PRFs in NC^1 secure against P/poly), though our new assumption is somewhat stronger and less justified than theirs.

Applications. Despite some minor 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 polynomial number of users. This is the

[4] A previous version of this work constructed constant-degree maps. The construction in this version is the same as before, but we have upgraded the security analysis to work for polynomial degree.

most efficient scheme for $n > 3$ users that is immune to known attacks. Hopefully, our maps can be used to make other applications much more efficient as well.

Ideal Multilinear Maps from GGH13. We note that our techniques for constructing multilinear maps from CLT13 can be combined with the techniques of Garg et al. [34] to give new multilinear maps in the weak model for GGH13.

1.2 Techniques

Weak CLT13 Model In CLT13, there is a composite modulus $N = \prod_i p_i$. [5] 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. An element in 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.[6] The 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.

All known attacks on the CLT13 multilinear maps 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 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 $\{s^{(j)}\}_j$, where $s^{(j)}$ denotes the jth encoding of the jth plaintext vector.
- 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 $t = \sum_i \gamma_i p(\{s_i^{(j)}\}_j)$. Here, $s_i^{(j)}$ represents the ith component of $s^{(j)}$, and $\{s_i^{(j)}\}_j$ denotes the collection of ith components of the various encodings provided.[7]
- 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 $\{s_i^{(j)}\}_j$ and T is substituted for the set of t obtained above. We say the adversary wins if she 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 $\{s_i^{(j)}\}_j$. Then by taking $GCD(N, s - s_i)$ for some encoding s, she obtains the prime factor p_i.

[5] In [18], this modulus is referred to as x_0.

[6] In our model, we will actually have s_i denote the numerator of the ith component of an encoding, but we ignore this distinction for this overview.

[7] To avoid confusion with other superscripts later in this work, these $s_i^{(j)}$ terms are written as $s_{j,i}$ in Section 3.

The real adversary does this for every i until she completely factors N. In general, solving Q for $\{s_i^{(j)}\}_j$ 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 a "model conversion theorem" that implies any attack in our weak 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, $R(\{p(\{\widehat{e}_j\}_j)\}_p)$ is identically zero as a polynomial over $\{\widehat{e}_j\}_j$, where $\{\widehat{e}_j\}_j$ are generic formal variables.

With this simpler model in hand, we immediately obtain the VBB-security of existing obfuscation constructions [38, 29] 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 Un-Annihilatability (BPUA) Assumption of [34], we conclude that it is impossible to find an annihilating polynomial R, and hence a polynomial Q_i. This gives security in our weak CLT13 model.

Using Our Analysis: Some Examples Our model conversion theorem gives a simple way to test if a given set of CLT13 encodings may be vulnerable to zero-izing attacks. As a concrete example, consider the Cheon et al. [20] attack in the asymmetric setting when there are n primes and the top level is $\{1, 2, 3\}$. The attack can be mounted given n encodings $a_1, \ldots, a_n$ at level $\{1\}$, 2 encodings b_0, b_1 at level $\{2\}$, and n more encodings $c_1, \ldots, c_n$ at level $\{3\}$, along with the condition that $a_i b_j c_k$ is an encoding of zero at level $\{1, 2, 3\}$ for all choices of $i, k \in [n]$ and $j \in \{0, 1\}$ (this can be achieved, for example, if b_0 and b_1 are encodings of zero). In the real attack, the adversary begins by constructing all $2n^2$ encodings of the form $a_i b_j c_k$ and zero-testing each one of them.

The model conversion theorem essentially states the following: a set of encodings is potentially vulnerable to zeroizing attacks if the top-level encodings submitted for (successful) zero-tests, interpreted as polynomials that treat encodings as formal variables, can be annihilated.[8] This condition easily confirms the setting of the Cheon et al. [20] attack is insecure.

The set of polynomials that compute top level zero-encodings is $\{p_{i,j,k} = \widehat{a}_i \widehat{b}_j \widehat{c}_k\}_{i,k\in[n],j\in\{0,1\}}$ (where the hat symbol indicates we are treating the encodings as formal variables), and the following polynomial gives a non-trivial annihilation:

$$(\widehat{a}_1 \widehat{b}_0 \widehat{c}_1)(\widehat{a}_2 \widehat{b}_0 \widehat{c}_2) - (\widehat{a}_1 \widehat{b}_0 \widehat{c}_2)(\widehat{a}_2 \widehat{b}_0 \widehat{c}_1) \equiv 0.$$

Note that this annihilating polynomial would have been possible even without the b_1 encoding, which the original Cheon et al. [20] attack seems to require.

Our model conversion theorem demonstrates that given any Q representing a successful zeroizing attack, we can derive an annihilating polynomial such as the one above. Given this theorem, proving security to zeroizing attacks is straightforward; one simply needs to show that the polynomials corresponding to successful zero-tests cannot be annihilated. As a trivial example of a setting our analysis deems secure, let $a_1, \ldots, a_n$ be encodings of random plaintexts at level $\{1\}$, let b be a single encoding of zero at level $\{2\}$, and let the top level be $\{1, 2\}$. The only polynomials that can give successful zero tests, written with formal variables, are $\widehat{a}_1 \widehat{b}, \widehat{a}_2 \widehat{b}, \ldots, \widehat{a}_n \widehat{b}$. As these polynomials are algebraically independent, they cannot be annihilated.

We note that most interesting applications will yield polynomials that are *not* algebraically independent, meaning annihilating polynomials certainly exist. However, by relying on computational assumptions, we can sometimes show that computing such annihilating polynomials in computationally intractable.

[8] To prevent trivial annihilations, we require (without loss of generality) that the set of successful zero-tests be linearly independent as polynomials. Also, the annihilating polynomial must be given as a polynomial-size circuit.

New Multilinear Maps We now turn to developing a new multilinear map scheme that we can prove secure in our weak model for CLT13. Guided by our annihilation analysis, we design the scheme to only release encodings for which the successful zero-test polynomials cannot be annihilated by polynomial-size circuits; our model conversion theorem shows that such encodings will be secure in the weak CLT13 model.

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. It is straightforward to extend to symmetric multilinear maps. The scheme will be based heavily on obfuscation techniques, plus some new techniques that we develop; however, we 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, Gentry, Halevi, and Zhandry [6][9], which offered a potential fix to block zeroizing attacks that we call the GGHZ16 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:

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

where \$ represent plaintexts drawn at random[10], 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 [26], 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 GGHZ16 fix as follows: arrange the matrices in a grid where 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. One way to see this is through the lens of our model conversion theorem: 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

[9] We actually need the version of [6] 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.

[10] Actually, in [6], 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

partition the input bits into two sets, and select subsets S and 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 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. [34]'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 in the number of reads, 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 GGHZ16 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 a d-level meta-multilinear map, we will need $d\ell + 2$ CLT13 levels (the extra 2 levels for the bookends). An 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:

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

Essentially, each of our meta-encodings is a list of GGHZ16 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 choose different randomizing matrices R in each position; this corresponds to the randomizing matrices used for Kilian [40] 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 [1]. We explain how to give intermediate levels below.

The pairing operation takes one meta-encoding for each meta-level, and arranges 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, owing to most of the GGHZ16 meta-encodings containing 0. Instead, we add an offset vector midway through the pairing operation to make the CLT13 encoding an encoding of the correct value; see Section 4 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 are 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 [5, 41], 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. 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. [29] which tightly characterizes 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, 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, by our model conversion theorem, if the adversary can attack in our weak CLT13 model, it can find an annihilating polynomial for vector-input branching programs. We therefore formulate a concrete 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 can 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. We note that, if we were to ignore the first issue, it is straightforward to overcome the second issue using a sub-exponentially secure PRF, since a sub-exponential time algorithm can potentially query a PRF on the entire domain and construct exponential-sized linear combinations. Furthermore, we hope that this second limitation arises from our analysis and is not a fundamental problem, leaving room for subsequent work.

We observe that if a sub-exponentially secure PRF can be computed by vector-input branching programs, it should be possible to prove our assumption based on the security of said PRF.
Even without a justification based on general hardness assumptions, the only efficient annihilating polynomials we could find for vector-input branching programs are determinant polynomials as described above. These are interestingly 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 w^ℓ to be 2^λ for desired security parameter λ, this will block known attacks. In Appendix E, we give some evidence for why this conjecture should hold in restricted settings.

- Our discussion above only allows for directly pairing to the top level. If we are willing to sacrifice polynomial degree for constant degree, we can 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 the different Kilian randomization matrices between each pair of levels, we cannot directly pair non-adjacent levels, such as 1 and 3. For non-adjacent levels, instead of matrix multiplications, we can *tensor* the encodings, generating all degree 2 monomials. This tensoring can also be extended to higher levels. Unfortunately, this greatly expands the size of encodings and thus can only be done when the degree is constant.
  Alternatively, we note that the ability to only multiply adjacent levels corresponds to the "graph induced" multilinear map notion [19] for the line graph. Hence, we obtain a multilinear map for general line graphs. Such maps are sufficient for most applications. Moreover, such graphs can easily be used to build symmetric multilinear maps, by simply encoding at all possible singleton levels.
- Finally, 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 the 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 polynomial 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 [1] will not work because (1) they need an 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 [1].

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-randomization), *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 proven 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 vector-input branching program which

only takes a single input vector. 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. Consider a general linear combination of iterated matrix products. 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.[11]. 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 $[a]_S$.
- (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 $S_1 \cap S_2 = \emptyset$. The 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 of a potentially unknown element a, and outputs a "fresh" encoding of a, distributed statistically close to a direct encoding of a.

[11] This technically corresponds to our construction when the degree of multi-linearity is 1, but the idea readily extends to arbitrary degree

– (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, which the oracle responds to by returning a bit indicating whether or not the corresponding element is an encoding of zero at level $\mathbb{U}$. We give a more precise description of the generic model in Section 3.

2.2 Overview of the CLT13 Multilinear Maps

We give a brief overview of the CLT13 multilinear maps, adapted from text in [42]. For a full description of the scheme, see [18]. 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 primes 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 $p_{zt} = \sum_{k=1}^{n} p_k^* h_k ((\prod_{i \in [\kappa]} z_i) g_k^{-1} \bmod p_k) \pmod{N}$ in $\mathbb{Z}_N$ where $p_k^* = N/p_k$, and comparing the result to N. An encoding α can be expressed as $\frac{1}{\prod_{i=1}^{\kappa} z_i} \mathsf{CRT}(s_k)_k$ where s_k denotes the numerator of its kth CRT component. When α is an encoding of zero, it can be shown that

$$p_{zt}\alpha \pmod{N} = \sum_{k=1}^{n} \gamma_k s_k,$$

where $\gamma_k = p_k^* h_k g_k^{-1}$ are "smallish" global secret parameters that depend on the other CLT13 parameters. For encodings of zero, each s_k is "small", so $p_{zt}\alpha \pmod{N}$ is small relative to N. If α does not encode 0, then one heuristically expects $p_{zt}\alpha \pmod{N}$ to be large relative to N. Thus, we can zero test by determining if this quantity is small.[12] We can also extract a unique representation of any encoded element by computing $p_{zt}\alpha \pmod{N}$, and rounding appropriately.

2.3 Vector-Input Branching Programs

We generalize matrix branching programs to vector-input branching programs (VBPs), a new notion we define (for a formal definition of matrix branching programs, refer to Appendix A). Single-input branching programs consist of a sequence of pairs of matrices, where an input bit is read for each pair to select one

[12] 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 far less efficient, so most implementations only use a single zero test vector as described here.

of them. Our vector input generalization allows for selecting a *linear combination* of matrices. In addition, we replace pairs of matrices with sets of k matrices. Thus, program inputs consist of d non-zero vectors of dimension k.

These vector-input branching programs arise from the security proof of our new multilinear map construction in Section 4. Thus, we will only consider a restricted class of vector-input branching programs tailored to fit the requirements of our analysis. Specifically, we use read-ℓ programs, meaning that each vector in the input is read exactly ℓ times. These programs are single-input, so there are $d\ell$ sets of matrices. Furthermore, the input selection is fixed so that the ith input vector is read for matrix sets $i, d+i, \ldots, (\ell-1)d+i$. In other words, the input selection function is simply $\mathsf{inp}(j) = j \pmod d$.

Definition 1. *A read-ℓ vector-input branching program over a ring R with input length n, vector dimension k, and matrix width w is given by a sequence*

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

where each $\boldsymbol{B}_{i,j}$ is a $w \times w$ matrix, s is a w-dimensional row vector, and t is a w-dimensional column vector. All entries are elements in R. Then $VBP : (\mathbb{Z}^k)^d \to R$ is computed as

$$VBP(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_d) = s \cdot \left(\prod_{i=1}^{d\ell} \left(\sum_{j=1}^{k} x_{i(mod\ d),j} \boldsymbol{B}_{i,j} \right) \right) \cdot t.$$

where $x_{i(mod\ d),j}$ denotes the jth component of the k-dimensional integer vector $\boldsymbol{x}_{i(mod\ d)}$, and $i(mod\ d)$ is shorthand for the integer in $\{1, \ldots, d\}$ equivalent to i modulo d.

We will frequently consider *generic vector-input branching programs*, meaning programs where each matrix in the program has entries that are all distinct formal variables. For examples and intuition of how to evaluate VBPs and generic VBPs, refer to Section 6

2.4 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 [40] 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 The Model

In this section, we define two models. The first is the weak CLT13 model, intended to capture all known classical attacks on the CLT13 multilinear maps. The second is the CLT13 annihilation model, a modification of the weak CLT13 model with different winning conditions. We justify our first model by demonstrating that it captures known attacks from the literature. 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. Combining this theorem with the Branching Program Un-Annihilatability Assumption of [34], we immediately obtain virtual black box (VBB) security of the obfuscator of Badrinarayanan et al. [29]. Additionally, this shows that the order-revealing encryption scheme of Boneh et al. [43] is secure in our model.

3.1 CLT13 Weak Multilinear Map Model

Notation. In certain settings, variables will represent actual values (such as ring elements or integers), and in other settings they will be used as formal variables. To differentiate these use cases, plain lowercase letters such as m, s, t, γ will denote actual values, while $\widehat{m}, \widehat{s}, \widehat{t}, \widehat{\gamma}$ should be viewed as formal variables. When we refer to "handles" for specific values m, s, t, γ, i.e. labels that point to these values, we will slightly abuse notation and denote these handles as $\widehat{m}, \widehat{s}, \widehat{t}, \widehat{\gamma}$. While handles and formal variables are technically distinct notions, there will be no meaningful difference between the two in this section. Bold letters will be used to distinguish vectors from scalars. For a positive integer D, let $[D]$ denote the set $\{1, 2, \ldots, D\}$.

We now define our weak CLT13 model, with 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. [18]. 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 D plaintext vectors $\{\mathbf{m}_j\}_{j\in[D]}$ where each $\mathbf{m}_j \in R_{\mathsf{ptxt}}$, as well as an encoding level S_j for each plaintext. For a plaintext vector $\mathbf{m}_j$, let $m_{j,i}$ denote the ith component, where $i \in [n]$. $\mathcal{M}$ generates the D vectors of CLT13 numerators $\{\mathbf{s}_j\}_{j\in[D]}$ where each $\mathbf{s}_j$ vector is generated component-wise by setting $s_{j,i} = m_{j,i} + g_i r_{ji}$ for each $i \in [n]$ as in the CLT13 encoding procedure. For each j, $\mathcal{M}$ stores the tuple $(\mathbf{m}_j, \mathbf{s}_j, S_j)$ in a pre-zero test table.

Zero-testing. The adversary submits a polynomial p_u to $\mathcal{M}$, represented as a polynomial-size *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 $p_u(\{m_{j,i}\}_{j\in[D]}) = 0$ for all i. If the check fails for any i, $\mathcal{M}$ returns "fail". If the check passes for all i, $\mathcal{M}$ returns "success". We assume without loss of generality that the set $\{p_u\}_u$ 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 [41]. 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 $\widehat{t}_u$ to the value

$$t_u(\{\gamma_i\}_{i\in[n]}, \{s_{j,i}\}_{j\in[D],i\in[n]}) = \sum_{i\in[n]} \gamma_i p_u(\{s_{j,i}\}_{j\in[D]}),$$

the result of the zero-test computation. Each handle $\widehat{t}_u$ along with the corresponding the zero-test result is stored in a *zero-test table.*

To simplify notation, we will drop the "$\in [n]$" and "$\in [D]$" when it is clear from context.

Post-zero-test. Finally, the adversary submits a polynomial Q on the handles $\{\widehat{t}_u\}_u$ and the formal variables $\{s_{j,i'}\}_j$ for some $i' \in [n]$ (that $\mathcal{A}$ picks). This Q must be represented by a polynomial-sized algebraic circuit, and the degree must be at most $2^{o(\lambda)}$.[13] The model looks up each handle $\widehat{t}_u$ 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. $Q(\{t_u\}_u, \{\widehat{s}_{j,i'}\}_j) \not\equiv 0$ as a polynomial over formal variables $\{\widehat{s}_{j,i'}\}_j$ for i' that $\mathcal{A}$ picked.
2. $Q(\{t_u\}_u, \{s_{j,i'}\}_j) = 0$.

[13] We note that these restrictions are analogous to restrictions made for annihilation attacks on GGH13 [34]

Recall that each t_u is the value $\sum_i \gamma_i p_u(\{s_{j,i}\}_{j,i})$ and can be viewed as a polynomial $t_u(\{\gamma_i\}_i, \{s_{j,i}\}_{j,i})$.

Intuitively, these conditions imply that Q is a polynomial with non-zero degree over the $\{\widehat{s}_{j,i'}\}_j$ formal variables that is "solved" when the correct values $\{s_{j,i'}\}_j$ are plugged in.

Plain Annihilation Model. We define a modification 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 $\{\widehat{p}_u\}_u$, where $\widehat{p}_u$ represents the successful zero test polynomial p_u. Again, this Q' must be represented by a polynomial-sized algebraic circuit, and the degree must be at most $2^{o(\lambda)}$. The model outputs "WIN" if the following conditions are satisfied.

1. $Q'(\{\widehat{p}_u\}_u)$ *is not* identically zero over the $\{\widehat{p}_u\}_u$ formal variables.
2. $Q'(\{p_u(\{\widehat{e}_j\}_{j\in[D]})\}_u)$ *is* identically zero over the $\{\widehat{e}_{j\in[D]}\}_j$ formal variables.

In other words, $\mathcal{A}$ wins if it submits a Q' that annihilates the $\{p_u\}_u$ polynomials.

Observe that condition 2 simply states that each p_u, a polynomial over D formal variables, is identically zero over these formal variables. We use a new formal variable letter $\widehat{e}_j$ for full generality, though in our analysis these $\widehat{e}_j$ will correspond to the formal variables $\widehat{s}_{j,k}$ for some choice of k.

3.2 Classical Attacks in the Weak CLT13 Model

We first show that the original attack on the CLT13 multilinear maps by Cheon et al. fits into this framework [20].

Mounting this attack requires that the set of plaintext vectors $\{\mathbf{m}_j\}$ given to $\mathcal{M}$ 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 clarity of exposition, we depart slightly from our generic indexing notation and label 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 vector 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 which evaluate to

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

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

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

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. [20] show has eigenvalues $\frac{s_{1,i}^B}{s_{2,i}^B}$ for $i \in [n]$. 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 $\widehat{t}$, and can only be solved for ratios of the $s_{j,i}$ values. However, we observe that the characteristic polynomial

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

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

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

$\mathcal{A}$ submits the left-hand side expression above as its polynomial Q in a post-zero-test query (note in this attack that Q will result in a win for any choice of i, though our model allows polynomials Q that only result in a win for some fixed i'). Since the Cheon et al. attack is successful (i.e. gives non-trivial solutions when solving for $\{s^B_{\sigma,i}\}_\sigma$), we know Q is nonzero over the formal variables $\{\widehat{s}^B_{\sigma,i}\}_\sigma$ after the zero-test results $\{t_{j,\sigma,k}\}$ are plugged in. Additionally, plugging in the correct numerator values $\{s^B_{\sigma,i}\}_\sigma$ satisfies the above expression, so both win conditions are satisfied. Thus, $\mathcal{A}$ wins in our model.

In Appendix B, we show how the general attack framework of Coron et al. [26] can be expressed in our model.

3.3 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.*

The proof strategy is as follows. An adversary that wins in the weak CLT13 model produces a non-trivial polynomial Q that evaluates to 0 on the actual CLT13 parameters and the numerators of the encodings. Since the CLT13 parameters and encodings are sampled using randomness hidden from the adversary, we can use a generalization of the Schwartz-Zippel lemma to conclude that the polynomial must be identically zero over its formal variables. We can view this polynomial as being over the formal variables corresponding to the CLT13 parameters, where the remaining formal variables constitute "coefficients". Since the overall polynomial is identically zero, all coefficients must also be identically zero. To conclude, we use the fact that the polynomial is non-trivial to show that there must exist a coefficient which acts as an annihilating polynomial for the zero-test polynomials.

Proof. Suppose $\mathcal{A}$ produces a post-zero-test polynomial Q and some $i' \in [n]$ satisfying the two win conditions:

$$Q(\{t_u(\{\gamma_i\}_i, \{s_{j,i}\}_{j,i})\}_u, \{\widehat{s}_{j,i'}\}_j) \not\equiv 0$$
$$Q(\{t_u(\{\gamma_i\}_i, \{s_{j,i}\}_{j,i})\}_u, \{s_{j,i'}\}_j) = 0$$

We claim that the second condition implies

$$Q(\{t_u(\{\widehat{\gamma}_i\}_i, \{\widehat{s}_{j,i}\}_{j,i})\}_u, \{\widehat{s}_{j,i'}\}_j) \equiv 0$$

with overwhelming probability. In this polynomial,

$$t_u(\{\widehat{\gamma}_i\}_i, \{\widehat{s}_{j,i}\}_{j,i}) = \sum_{i\in[n]} \widehat{\gamma}_i p_u(\{\widehat{s}_{j,i}\}_j).$$

In words, this means that instead of computing t_u from the values of γ_i and $s_{j,i}$, we substitute formal variables $\widehat{\gamma}_i$ and $\widehat{s}_{j,i}$ into the definition of t_u, resulting in a polynomial over these variables.

Suppose for the sake of contradiction that

$$Q(\{t_u(\{\widehat{\gamma}_i\}_i, \{\widehat{s}_{j,i}\}_{j,i})\}_u, \{\widehat{s}_{j,i'}\}_j) \not\equiv 0$$

Using a variant of the Schwartz-Zippel lemma due to Garg et al. [34], we can show that with all but negligible probability, this would violate the second win condition. Essentially, the lemma states that if a polynomial over random variables is not identically zero and the variables have sufficient min entropy relative to each other, then the polynomial will not evaluate to zero with overwhelming probability (see Appendix D for a full statement of this lemma). Since $s_{j,i} = a_{j,i} + g_i r_{j,i}$ and each $r_{j,i}$ is sampled independently from a high entropy distribution, the $s_{j,i}$'s have significant min-entropy conditioned on the other $s_{j,i}$ and γ_i variables. The γ_i variables have significant min-entropy since $\gamma_i = \frac{h_i N}{p_i g_i}$, and the h_i's are randomly and independently sampled $\Omega(\lambda)$-bit integers.

Now we expand the polynomial $Q(\{t_u(\{\widehat{\gamma}_i\}_i, \{\widehat{s}_{j,i}\}_{j,i})\}_u, \{\widehat{s}_{j,i'}\}_j)$ in terms of its formal variables. Since each $t_u(\{\widehat{\gamma}_i\}_i, \{\widehat{s}_{j,i}\}_{j,i})$ is a linear polynomial in the $\widehat{\gamma}_i$ variables, a monomial with total degree d_t on the t_u terms can only be canceled out by other monomials with the same total degree on the t_u terms. Thus without loss of generality we restrict our attention to polynomials Q that can be written as a sum of terms where each term has a total degree of d_t on the t_u terms.

So we write Q in the form

$$\sum_{|S|=d_t} a_S t_{S_1} t_{S_2} \cdots t_{S_{d_t}} R_S(\{\widehat{s}_{j,i'}\}_j),$$

where S runs over all multi-sets of size d_t of elements from $[U]$ (where U is such that $\{p_u\}_{u\in[U]}$ is the set of successful zero-test polynomials submitted by the adversary), and we denote the elements of S as $S_1, S_2, \ldots, S_{d_t}$. The expression on the right-hand side can be expanded as

$$\sum_{|S|=d_t} a_S \left(\sum_i \widehat{\gamma}_i p_{S_1}(\{\widehat{s}_{j,i}\}_j)\right)\left(\sum_i \widehat{\gamma}_i p_{S_2}(\{\widehat{s}_{j,i}\}_j)\right)\cdots\left(\sum_i \widehat{\gamma}_i p_{S_{d_t}}(\{\widehat{s}_{j,i}\}_j)\right) R_S(\{\widehat{s}_{j,i'}\}_j).$$

The coefficient of $\widehat{\gamma}_k^{d_t}$ for any $k \neq i'$ is

$$\sum_{|S|=d_t} a_S \left(p_{S_1}(\{\widehat{s}_{j,k}\}_j)\right)\left(p_{S_2}(\{\widehat{s}_{j,k}\}_j)\right)\cdots\left(p_{S_{d_t}}(\{\widehat{s}_{j,k}\}_j)\right) R_S(\{\widehat{s}_{j,i'}\}_j),$$

which we note must be identically zero as this monomial cannot be canceled out by any other terms. Observe that since Q was written in the form

$$\sum_{|S|=d_t} a_S t_{S_1} t_{S_2} \cdots t_{S_{d_t}} R_S(\{\widehat{s}_{j,i'}\}_j) = Q(\{t_u(\{\widehat{\gamma}_i\}_i, \{\widehat{s}_{j,i}\}_{j,i})\}_u, \{\widehat{s}_{j,i'}\}_j)$$

the coefficient of $\widehat{\gamma}_k^{d_t}$ can be re-expressed as

$$\sum_{|S|=d_t} a_S \left(p_{S_1}(\{\widehat{s}_{j,k}\}_j)\right)\left(p_{S_2}(\{\widehat{s}_{j,k}\}_j)\right)\cdots\left(p_{S_{d_t}}(\{\widehat{s}_{j,k}\}_j)\right) R_S(\{\widehat{s}_{j,i'}\}_j) = Q(\{p_u(\{\widehat{s}_{j,k}\}_j)\}_u, \{\widehat{s}_{j,i'}\}_j).$$

Recall that $Q(\{t_u(\{\widehat{\gamma}_i\}_i, \{\widehat{s}_{j,i}\}_{j,i})\}_u, \{\widehat{s}_{j,i'}\}_j)$ is an identically zero polynomial over all of its formal variables. Thus, $Q(\{p_u(\{\widehat{s}_{j,k}\}_j)\}_u, \{\widehat{s}_{j,i'}\}_j)$, which is the "coefficient" of the $\widehat{\gamma}_k^{d_t}$ term in the polynomial $Q(\{t_u(\{\widehat{\gamma}_i\}_i, \{\widehat{s}_{j,i}\}_{j,i})\}_u, \{\widehat{s}_{j,i'}\}_j)$, must also be identically zero over its formal variables.

Now we apply this technique again to extract a coefficient of a term in $Q(\{p_u(\{\widehat{s}_{j,k}\}_j)\}_u, \{\widehat{s}_{j,i'}\}_j)$ that must be identically zero. In particular, view the $\widehat{s}_{j,i'}$ terms as the variables of this polynomial and everything else as comprising the coefficients. All of the coefficients are then polynomials over the $\{\widehat{s}_{j,k}\}_j$, and we can conclude these coefficient polynomials must all be identically zero since the $\{\widehat{s}_{j,k}\}_j$ formal variables are disjoint from the $\{\widehat{s}_{j,i'}\}_j$ formal variables. We write this polynomial as $\sum_V c_V(\{p_u(\{\widehat{s}_{j,k}\}_j)\}_u)\prod_{j\in V}\widehat{s}_{j,i'}$ where V runs over multisets of indices $j \in [n]$, and in this notation $c_V(\{p_u(\{\widehat{s}_{j,k}\}_j)\}_u) \equiv 0$ for all V.

Now we invoke the first condition, which states $Q(\{t_u(\{\gamma_i\}_i, \{s_{j,i}\}_{j,i})\}_u, \{\widehat{s}_{j,i'}\}_j) \not\equiv 0$. Replacing each $t_u(\{\gamma_i\}_i, \{s_{j,i}\}_{j,i})$ value with a distinct formal variable $\widehat{t}_u$ cannot make the polynomial identically zero. Thus, $Q(\{\widehat{t}_u\}_u, \{\widehat{s}_{j,i'}\}_j) \not\equiv 0$. We can write $Q(\{\widehat{t}_u\}_u, \{\widehat{s}_{j,i'}\}_j)$ as a polynomial over the $\widehat{s}_{j,i'}$ formal variables with coefficients in terms of the T_u formal variables. The previous paragraph implies that each of these coefficients becomes identically zero when evaluated on the $\{p_u(\{\widehat{s}_{j,k}\}_j)\}_u$ polynomials. Since Q is a nonzero polynomial, there exists a coefficient c_V that is not identically zero. Thus, c_V is a non-trivial annihilating polynomial for $\{p_u(\{\widehat{s}_{j,k}\}_j)\}_u$, and an adversary $\mathcal{A}'$ can submit c_V to win in the plain annihilation model. To match the notation of the win condition in the plain annihilation model, we can set the formal variables $\widehat{e}_j = \widehat{s}_{j,k}$.

3.4 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. [29] show for their obfuscator (which is a tweak of the obfuscator of Barak et al. [38]), 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 [29], 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. [34], we immediately conclude that no such annihilating polynomial is possible. Thus, there is no weak CLT13 attack on this obfuscator.[14]

We similarly observe that in the order-revealing encryption (ORE) scheme of Boneh et al. [43], 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.

4 A New Multilinear Map Candidate

In this section, we give a candidate polynomial-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. Here, we discuss our basic scheme; in Section 7, we give several optimizations.

4.1 Construction Overview

The levels will be non-empty subsets of $[d]$ for some polynomial 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 [1].

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 $u, u+d, u+2d, \ldots, u+(\ell-1)d$. 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 $A_i^{(u)}$ the ith matrix in the encoding for level u. To construct $A_i^{(u)}$, we first define the diagonal matrix $\widetilde{A_i^{(u)}}$, of the form

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

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 d block matrices, where $d-1$ blocks are essentially unused and set to random multiples of the identity, while the uth matrix is set to be an "enforcing matrix" $E_i^{(u)}$. Note that i corresponds to this being the ith matrix for encoding u. The purpose of these matrices is roughly to prevent an adversary from arbitrarily mixing and matching the matrices from different encodings. We defer the details of these matrices to Section 4.2.

[14] The Garg et al. obfuscator is defined as a dual-input obfuscator, which is the version we consider. The dual-input requirement is crucial; a single-input variant of this obfuscator is insecure and was attacked by Coron et al. [44]. The key point is that the branching program un-annihilatability assumption only holds for branching programs with significant interleaving of input bits, which can be ensured by a dual-input requirement.

Next, $d\ell + 1$ Kilian randomization matrices R_i are generated [40] to left and right multiply each of the $d\ell$ columns. All encodings will share the same Kilian matrices. Each $\widetilde{A_i^{(u)}}$ matrix at meta-level u is left- and right- multiplied by the appropriate Kilian matrices, giving

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

Each element of this Kilian-randomized 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 $A_i^{(u)}$.

For a matrix $A_i^{(u)}$, we refer to the underlying ring element 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 4.2. 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 $(A_i^{(u)})_{i\in[\ell]}$ 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 4.2, 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 4.2 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" [45]. 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.

4.2 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 $\{u\}$, which all 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. As described in Section 4.1, the rest of the diagonal entries are divided into d equally-sized diagonal matrices. The uth block is set to $E_i^{(u)}$, which provides the enforcing structure for level $\{u\}$, while the other $d-1$ blocks are set to random multiples of the identity to avoid interfering with the enforcing structure of the other singleton levels.

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

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

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 $2(\ell-1)$. To set the entries of G, 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 $i \in [\ell-1]$.

Restrictions on Matrix Products The sole purpose of the enforcing matrices is to ensure that the adversary respects the meta-encoding structure. For example, since each meta-encoding consists of ℓ separate matrices, an adversary may try to swap some of these matrices for matrices from other meta-encodings. We show that any attempt to do so will inevitably lead to a useless top-level encoding of a random plaintext.

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 $k \times d\ell$ grid, with k matrices in each of the $d\ell$ columns. Since we have k encodings per singleton level, we modify our notation slightly; now $A_{i,j}^{(u)}$ will denote the jth encoding of the ith matrix for meta-level u. Furthermore, we can ignore the first three rows

and columns of each $A_{i,j}^{(u)}$ matrix, as they play no role in the enforcing structure. Let $C_{i,j}^{(u)}$ be the width $2d(\ell-1)$ diagonal matrix that remains.

To recap, $d-1$ of the blocks of $C_{i,j}^{(u)}$ are set to be width-$2(\ell-1)$ identity matrices (randomly scaled) and the uth block is set to $E_{i,j}^{(u)}$. For any meta-encoding j at level u, a fresh random set of $\{E_{i,j}^{(u)}\}_{i\in[\ell]}$ is generated. The bookends are formed by concatenating d independently generated instances of our $2(\ell-1)$ dimensional bookends. The arrangement of the $C_{i,j}^{(u)}$ matrices (without the bookends) in the $k \times d\ell$ grid is shown below:

$$\left|\begin{array}{ccc|ccc|c|ccc} C_{1,1}^{(1)} & C_{1,1}^{(2)} & \cdots & C_{1,1}^{(d)} & C_{2,1}^{(1)} & C_{2,1}^{(2)} & \cdots & C_{2,1}^{(d)} & & C_{\ell,1}^{(1)} & C_{\ell,1}^{(2)} & \cdots & C_{\ell,1}^{(d)} \\ C_{1,2}^{(1)} & C_{1,2}^{(2)} & \cdots & C_{1,2}^{(d)} & C_{2,2}^{(1)} & C_{2,2}^{(2)} & \cdots & C_{2,2}^{(d)} & \cdots & C_{\ell,2}^{(1)} & C_{\ell,2}^{(2)} & \cdots & C_{\ell,2}^{(d)} \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots & & \vdots & \vdots & & \vdots \\ C_{1,k}^{(1)} & C_{1,k}^{(2)} & \cdots & C_{1,k}^{(d)} & C_{2,k}^{(1)} & C_{2,k}^{(2)} & \cdots & C_{2,k}^{(d)} & & C_{\ell,k}^{(1)} & C_{\ell,k}^{(2)} & \cdots & C_{\ell,k}^{(d)} \end{array}\right|$$

The matrices are divided into ℓ groups, each consisting of k rows and d columns of matrices. Picking the matrix in row j and column u for each group gives the ℓ matrices that comprise the enforcing component of the jth meta-encoding at level u.

Notice that adding point-wise $C_{1,j_0}^{(u)},\ldots,C_{\ell,j_0}^{(u)}$ to $C_{1,j_1}^{(u)},\ldots,C_{\ell,j_1}^{(u)}$ or scaling $C_{1,j}^{(u)},\ldots,C_{\ell,j}^{(u)}$ preserves the form of the matrices (though now the α's are different). Notice also that multiplying all the matrices in $C_{1,j}^{(u)},\ldots,C_{\ell,j}^{(u)}$ together along with our bookend vectors gives 0. Therefore, we can take arbitrary linear combinations of meta-encodings, multiply them together, and still get 0. We will show that this is essentially the only way to combine different $C_{i,j}^{(u)}$ to get zero.

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.

While the CLT13 level structure allows an adversary to multiply together any collection of matrices that picks one from each column, our enforcing matrix structure will restrict the adversary to taking products of linear combinations of meta-encodings (or linear combinations of such products). In fact, our analysis will limit these linear combinations to be very restricted vectors of non-negative integers.

Definition 2. *Let $H_{\ell,k}$ denote the set of k dimensional vectors of non-negative integers that sum to exactly ℓ. For any sequence of d vectors $x_1,\ldots,x_d \in H_{\ell,k}$, define the associated integer combination product as follows. The vector x_u specifies coefficients of a linear combination of the k level-u meta-encodings. Taking these linear combinations for $x_1,\ldots,x_d$ produces d meta-encodings. The integer combination is the product of these meta-encodings.*

Definition 3. *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.*

Our analysis will first show that our enforcing matrices enforce a property we call permutation invariance. Note to the reader: for full generality, the following definition and analysis will need to handle arbitrary levels of multilinearity d. However, the analysis readily extends to higher d, and focusing on the $d=1$ case suffices for understanding the techniques.

Definition 4. *Let Q be a general linear combination of iterated matrix products. Suppose (momentarily) that the degree of multi-linearity $d=1$. $v^{(1)}$ is a vector of the form $(v_1^{(1)},v_2^{(1)},\ldots,v_\ell^{(1)})$ where ℓ is the number of matrices in each meta-encoding. Let $\beta_{v^{(1)}} = \beta_{v_1^{(1)},v_2^{(1)},\ldots,v_\ell^{(1)}}$ be the coefficient of the linear combination corresponding to the iterated matrix product that selects the $v_i^{(1)}$th matrix from the ith column. Then for any*

permutation σ on $[\ell]$, Q is a permutation invariant polynomial if $\beta_{v_1^{(1)},v_2^{(1)},\ldots,v_\ell^{(1)}} = \beta_{v_{\sigma(1)}^{(1)},v_{\sigma(2)}^{(1)},\ldots,v_{\sigma(\ell)}^{(1)}}$. More generally, for arbitrary d, $\beta_{v^{(1)},\ldots,v^{(d)}}$ is the coefficient of the iterated matrix product where each $v^{(u)}$ is an ℓ-dimensional vector with entries in $[k]$ indicating which matrices are picked in columns $u, u+d, \ldots, u+(\ell-1)d$. Q is permutation invariant if, for any choices of d permutations $\sigma^{(1)}, \ldots, \sigma^{(d)}$ on $[\ell]$, $\beta_{v^{(1)},\ldots,v^{(d)}} = \beta_{\sigma^{(1)}(v^{(1)}),\ldots,\sigma^{(d)}(v^{(d)})}$, where $\sigma^{(u)}(v^{(u)})$ denotes the vector $v^{(u)}_{\sigma^{(u)}(1)}, \ldots, v^{(u)}_{\sigma^{(u)}(\ell)}$.

Once permutation invariance is established, we show that permutation invariance implies that Q is a linear combination of integer combination products.

Lemma 1. *Let Q be a linear combination of valid monomials. If Q evaluates to 0 as a polynomial over the underlying randomness of the $C^{(u)}_{i,j}$ matrices, then Q must permutation invariant. Furthermore, if permutation invariance holds, Q is a linear combination of integer combination products.*

Proof. First, we prove 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 $C^{(u)}_{(j)}$.

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 $\binom{\ell+k-1}{k-1} \times \binom{\ell+k-1}{k-1}$ 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 $k \geq \ell$ [46]. 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

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

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

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

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

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

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.

We note that integer combination products (Definition 2) correspond to an artificially restricted set of valid evaluations of our multilinear map construction. In particular, an integer combination product results from adding exactly ℓ meta-encodings (allowing repetition) at each level of multi-linearity, and then multiplying the results together. Clearly, even valid evaluations that respect meta-encoding structure need not be integer combination products. Thus, this decomposition analysis may seem strange at first glance. However, we note that any valid evaluation that respects meta-encoding structure will be permutation invariant. The above analysis then shows that any permutation invariant polynomial can be written as a linear combination of (potentially exponentially-many) integer combination products.

5 Security of Our Multilinear Map

Strategy Overview To prove the security of our multilinear map construction within the CLT13 weak model, we define "real" and "ideal" experiments. Recall that an encoding of a plaintext in our scheme consists of numerous CLT13 plaintexts at different CLT13 levels. The "real" experiment allows the adversary to perform operations on any of these individual CLT13 encodings, and win through any of the victory conditions in the weak model. The "ideal" experiment provides the same interface to the adversary, but is run by a simulator that only has access to the vanilla generic multilinear map model. Intuitively, if there exists a simulator for which the adversary cannot distinguish between the two experiments, then no extra information is leaked in the real world and our multilinear map achieves ideal security.

Real and Ideal Experiments 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 experiment, denoted $\texttt{EXP}_{real}$, 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 win by submitting a polynomial Q that satisfies the win conditions of the CLT13 weak model.

In the "ideal" world experiment, denoted $\texttt{EXP}_{ideal}$, 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.

For any adversary $\mathcal{A}$, let $\texttt{EXP}_{real}(\mathcal{A})$ (respectively $\texttt{EXP}_{ideal}(\mathcal{A})$) denote the probability that $\mathcal{A}$ can win in the "real" (respectively "ideal") world.

5.1 The Vector-Input Branching Program Un-Annihilatability (VBPUA) Assumption

The security of our multilinear map rests on a new assumption about annihilating vector-input branching programs (VBPs),defined in Section 2.3. Define a *generic* vector-input branching program (VBP) to be a VBP whose matrix entries are all distinct formal variables, instead of fixed ring elements. A generic VBP is evaluated just like a regular VBP, but the program output is a polynomial over formal variables. We refer to these outputs as generic VBP evaluation polynomials.

Note to the reader: The following is the simplest formal statement of our assumption. As the assumption relies on a number of newly introduced terms, it may be somewhat difficult to parse. In Section 6 we give concrete, intuitive examples of what our assumption statement means.

We now define the polynomial-size arithmetic circuits A_r that will be necessary for the assumption statement. A_r takes the matrices of the vector-input branching program as input. The output of A_r is restricted to be a linear combination of vector-input branching program evaluations (though note that a polynomial-size arithmetic circuit can compute exponentially large linear combinations).

Assumption 1 (The (ℓ, w, k)-VBPUA Assumption) *Let $\ell = \mathsf{poly}(d, \lambda), w = \mathsf{poly}(d, \lambda), k = \mathsf{poly}(d, \lambda)$ be parameters, and let $f(x)$ for $(x_1, \ldots, x_d) \in (\mathbb{Z}^k)^d$ (where each input vector x_i is a k-dimensional integer vector) be a generic vector-input branching program that reads each input vector x_i ℓ times and consists of $w \times w$ matrices. For each $r = 1, \ldots, m$, let A_r be any arithmetic circuits satisfying the definition above, each with size $\mathsf{poly}(d, k, \lambda)$. The output of A_r is a polynomial over the formal variables of the generic vector-input branching program, and denote it by P_r. Suppose further that $P_1, \ldots, P_m$ are linearly independent as polynomials over the formal variables of the generic vector-input branching program. 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.*

Note that each pair of functions ℓ, w, k gives a distinct assumption. In general, increasing ℓ or w intuitively gives a harder problem (and therefore milder assumption), while increasing k gives a potentially easier problem (and therefore stronger assumption. Our construction is quite flexible, an can be tailored to work with multiple possible settings of ℓ, w, k. Importantly, however, we will usually need k to be substantially larger than $w^2\ell$. For more precise bounds on ℓ, w, k, see Section 7.

We conjecture that the assumption is for any choices of ℓ, w, k provided w^ℓ is exponential in λ, d. We conjecture that determinant-style annihilating attacks (discussed in Section) give the lowest-complexity annihilating polynomials for VBPs, as this appears to be the case for matrix branching programs. w^ℓ lower bounds the size of such determinant-style annihilations. Verifying this would imply security for the choices of ℓ, w, k that we require.

This assumption is similar in spirit to the Branching Program Un-Annihilatability Assumption of Garg et al. [34], but we do not know how to base our assumption on PRFs. For a comparison of the two assumptions, see Section 6.

Note that the assumption states that no polynomial-size circuits Q of degree at most $2^{o(\lambda)}$ can annihilate a non-trivial set of VBP evaluations (generated by polynomial-size circuits). As evidence that this assumption is not trivially false, we heuristically argue in Appendix E that there are no circuits Q that can annihilate VBP evaluations up to a certain polynomial degree, even if we allow Q to have exponential size.

5.2 Security Proof

Our 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 PPT simulator S such that for all PPT adversaries $\mathcal{A}$,*

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

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, w, k)-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 $\{d\ell + 1\}$ respectively), and that the top level of the map is $\{0, 1, \ldots, d\ell + 1\}$. This means each monomial is a product of $d\ell + 2$ matrix entries, such that exactly one entry is chosen from each of the $d\ell$ columns, plus two entries chosen from the bookend vectors. Note that there may be exponentially many monomials in this polynomial.

We claim that if Q' is a successful annihilating polynomial over the matrix entries, it must in fact be a linear combination of matrix products. This follows from an application of Lemma 2, found in Appendix C. The collection of matrices $\{A_{i,j}\}$ in the statement of Lemma 2 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 apply Lemma 1, from which we conclude that Q' is a linear combination of integer combination products. Recall that integer combination products (introduced in Definition 2) are products of meta-encodings generated by summing an integer number of other meta-encodings (additionally, these integers must sum to ℓ).

We can interpret the public parameters of our 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 $n = d$ vectors in $H_{\ell,k}$ (the set of k dimensional vectors of non-negative integers summing to ℓ). We invoke the (ℓ, w, k)-VBPUA assumption with n set to d, $w = 3 + 2n(\ell - 1)$, and $\ell = \lambda / \log(w)$ (to block determinant-style attacks), and k to be a sufficiently large polynomial, as derived in Section 7. The assumption states that there does not exist a polynomial that can annihilate a linear combination of VBP evaluations on inputs $x \in H^n_{\ell,k}$, 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 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 p (represented as a polynomial-size arithmetic circuit) to S for zero-testing. This polynomial p 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 $m_i^{(u)}$.

If p is a successful zero-test polynomial, it must be a linear combination of matrix products due to Lemma 2 in Appendix C. Thus, the simulator first checks if p, a polynomial over the entries of matrices of encodings, is actually a linear combination of iterated matrix products. To do this, the simulator constructs an alternative circuit p_{matrix} which is structurally the same as p, except each input is a matrix (or one of the two bookend vectors). Addition gates are replaced with matrix addition gates, and multiplication gates are replaced with matrix multiplication gates. Whenever p takes as input the top left entry of some matrix, p_{matrix} takes as input that entire corresponding matrix. For inputs to p that are not the top left entry of some matrix, p_{matrix} takes in a 0 matrix. p and p_{matrix} compute the same functionality if and only if p is a linear combination of iterated matrix products. To check the equivalence of p and p_{matrix}, the simulator simply applies polynomial identity testing. The simulator samples random matrices (picking the entries from a large enough field) and checks if the results are the same. If the results are different, then p is not a linear combination of matrix products and the simulator returns "non-zero". Note that if p is not an iterated matrix product, p and p_{matrix} will be different the simulator catches this with high probability due to the Schwartz-Zippel Lemma.

The next step is for $\mathcal{S}$ to verify that p is in fact a linear combination of VBP evaluations. Due to the enforcing matrices, Lemma 1 implies that p must be of this form or the zero test will not be successful with high probability. To test this, we turn to the Lemma 1, which implies that p is a linear combination of VBP evaluations if and only if it is permutation invariant (the lemma shows permutation invariance of a polynomial Q implies it is a linear combination of specific VBP evaluations called integer combination products, and any VBP evaluation is permutation invariant).

We can check permutation invariance as follows. View p_{matrix} as a circuit over scalar inputs, which simply requires replacing matrix addition and multiplication gates with ordinary addition and multiplication gates. Call the resulting circuit p_{scalar}. For each $u = 1, \ldots, d$, we verify permutation invariance. This verification step is completely analogous to how our enforcing matrices ensure permutation variance and correctness is analogous to the proof of Lemma 1. We use the fact that when p_{scalar} is viewed as a scalar polynomial, permutation invariance for each choice of u can be checked by swapping the first and second columns corresponding to level u, then the second and third columns, etc. Note that we switch to a scalar polynomial to avoid issues that arise when swapping columns, as matrices do not commute. Recall that there are ℓ columns corresponding to each u, and it suffices to check $\ell - 1$ different transpositions. Each check is performed by sampling one random scalar for each matrix, plugging them into p_{scalar}, and evaluating the circuit. Then the random values assigned to two columns are swapped, and we evaluate p_{scalar} again and compare the values. Since the permutation invariant polynomials are precisely those polynomials that are invariant under permutations of the columns corresponding to each u, and the transpositions checked above generate precisely this set of permutations, with overwhelming probability (due to Schwartz-Zippel) these checks pass if and only if p_{scalar} is a permutation invariant polynomial.

So if all of these $(\ell - 1)d$ transposition checks pass, we conclude that p is permutation invariant, and if any check fails, $\mathcal{S}$ returns "non-zero."

At this stage, we know that p is a top-level encoding that respects the meta-encoding structure. Thus, $\mathcal{S}$ needs to query its multilinear map oracle to respond to this zero-test honestly. Notice that since p is a linear combination of VBP evaluations, the evaluation of p is identical to the evaluation of p_{scalar} on the matrix plaintexts (the top left entry of the pre-Kilian-randomized matrix before encoding), since everything else cancels out. Therefore, the simulator takes p_{scalar}, and for each matrix where $\mathcal{S}$ knows the matrix plaintext, $\mathcal{S}$ hardcodes in the value into p_{scalar}. Note that $\mathcal{S}$ knows the values of the top left entry of every matrix in every public parameter meta-encodings, as well as the top left entry (which is simply 0) for every matrix except the first matrix in all other meta-encodings. Let the resulting circuit be p'. Since p_{scalar} was a CLT13

level-respecting polynomial, p' derived in this way must be level-respecting. Therefore, $\mathcal{S}$ submits p' to its simulator as a zero-test query, and replies with the answer to the adversary.

6 Vector-Input Branching Programs and Our Assumption

In this section, we give explicit examples of how vector-input branching programs are evaluated. Using these examples we restate the Branching Program Un-Annihilatability Assumption of Garg et al. [34] and compare their assumption to our new Vector-Input Branching Program Un-Annihilatability Assumption. The purpose of this section is to give intuition for our new Vector-Input Branching Program notion and our new assumption.

6.1 Evaluating Generic Programs: Some Examples

Recall that we define a *generic* matrix branching program (MBP) or vector-input branching program (VBP) as one in which each matrix entry is left as a formal variable. Evaluating a generic MBP or VBP is done exactly the same as regular evaluation, except the result is a multivariate polynomial.

Also recall that branching programs are parameterized by the following parameters. We say that each layer of the branching program consists k different $w \times w$ matrices. The branching program reads an input of length d, and that each input digit is read ℓ times. We will assume the input selection function reads the first input digit in layers $1, 1+d, 1+2d, \ldots, 1+(\ell-1)d$, and the second input digit in layers $2, 2+d, 2+2d, \ldots, 2+(\ell-1)d$, etc.

We will assume each computation multiplies a $1 \times w$ dimensional bookend vector on the left and $w \times 1$ dimensional bookend vector on the right so that the result of computation is always a scalar.

As an example, consider the case where the matrix dimension $w = 2$, each layer contains $k = 2$ matrices, the length of the input is $d = 2$, and each input digit is read $\ell = 2$ times. We note that these parameters are unlikely sufficient for either the Branching Program Un-Annihilatability Assumption or the Vector-Input Branching Program Un-Annihililatability Assumption to hold, but we use them for the purpose of illustration.

The left bookend vector, the eight branching program matrices, and the right bookend vector are written below for a generic branching program of the above parameters:

$$\begin{bmatrix} S_1 & S_2 \end{bmatrix} \begin{matrix} \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} & \begin{bmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{bmatrix} & \begin{bmatrix} E_{1,1} & E_{1,2} \\ E_{2,1} & E_{2,2} \end{bmatrix} & \begin{bmatrix} G_{1,1} & G_{1,2} \\ G_{2,1} & G_{2,2} \end{bmatrix} \\ \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix} & \begin{bmatrix} D_{1,1} & D_{1,2} \\ D_{2,1} & D_{2,2} \end{bmatrix} & \begin{bmatrix} F_{1,1} & F_{1,2} \\ F_{2,1} & F_{2,2} \end{bmatrix} & \begin{bmatrix} H_{1,1} & H_{1,2} \\ H_{2,1} & H_{2,2} \end{bmatrix} \end{matrix} \begin{bmatrix} T_1 \\ T_2 \end{bmatrix}$$

Layer 1 Layer 2 Layer 3 Layer 4

Note that the input selection function specifies that the first input bit is used to select the matrix in layers 1 and 3, and the second input bit selects the matrix in layers 2 and 4.

For example, on input string "01", the first input bit 0 is read in layers 1 and 3, selecting the top matrix in those two layers, and the second input bit of 1 is read in layers 2 and 4, selecting the bottom matrix in those two layers. The result of the generic matrix branching program on input string "01" is then the scalar 20-variable polynomial that results from expanding out the following matrix product:

$$\begin{bmatrix} S_1 & S_2 \end{bmatrix} \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} \begin{bmatrix} D_{1,1} & D_{1,2} \\ D_{2,1} & D_{2,2} \end{bmatrix} \begin{bmatrix} E_{1,1} & E_{1,2} \\ E_{2,1} & E_{2,2} \end{bmatrix} \begin{bmatrix} H_{1,1} & H_{1,2} \\ H_{2,1} & H_{2,2} \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \end{bmatrix}$$

The above polynomial is referred to as the generic evaluation polynomial resulting from evaluating the generic branching program on the input 01, where $d = w = \ell = k = 2$.

These same matrices can be re-interpreted as a vector-input branching program. Now each input digit is replaced by a k-dimensional integer vector. The vectors are k-dimensional since there are k matrices in each layer, and each component of the vector specifies the integer coefficient on each matrix.

Since $k = 2$ in this example, a possible input to this program would be $\binom{4}{0}, \binom{2}{-1}$. The first input vector $\binom{4}{0}$ is read in layers 1 and 3, and is interpreted as a linear combination of the matrices where 4 is the coefficient on the first matrix and 0 is the coefficient on the second. The second input vector $\binom{2}{-1}$ is read in layers 2 and 4, and is interpreted as a linear combination of the matrices where 2 is the coefficient on the first matrix and -1 is the coefficient on the second.

Thus, the generic vector-input branching program evaluation is the scalar 28-variable polynomial that results from expanding out the following matrix product:

$$[S_1\ S_2]\left(4\begin{bmatrix}A_{1,1}\ A_{1,2}\\ A_{2,1}\ A_{2,2}\end{bmatrix}\right)\left(2\begin{bmatrix}C_{1,1}\ C_{1,2}\\ C_{2,1}\ C_{2,2}\end{bmatrix}-\begin{bmatrix}D_{1,1}\ D_{1,2}\\ D_{2,1}\ D_{2,2}\end{bmatrix}\right)\left(4\begin{bmatrix}E_{1,1}\ E_{1,2}\\ E_{2,1}\ E_{2,2}\end{bmatrix}\right)\left(2\begin{bmatrix}G_{1,1}\ G_{1,2}\\ G_{2,1}\ G_{2,2}\end{bmatrix}-\begin{bmatrix}H_{1,1}\ H_{1,2}\\ H_{2,1}\ H_{2,2}\end{bmatrix}\right)\begin{bmatrix}T_1\\ T_2\end{bmatrix}$$

6.2 The BPUA Assumption and the VBPUA Assumption

We compare our new assumption with the Branching-Program Un-Annihilatability Assumption of Garg et al. [34] for parameters d, w, ℓ, k can be re-interpreted as an assumption about annihilating generic matrix branching program evaluation polynomials.

BPUA Assumption. Suppose we have m linear combinations of matrix branching program evaluation polynomials, denoted as $P_1, \ldots, P_m$. To be concrete, each P_r is itself a linear combination of matrix branching program evaluation polynomials. The only requirement is that P_r be a linear combination of *polynomially-many* (in λ) matrix branching evaluations. Suppose furthermore that the $P_1, \ldots, P_m$ polynomials are linearly independent as polynomials over the variables of the generic branching program. Assumption: There does not exist a polynomial-size (in λ) circuit that computes an annihilating polynomial of degree $2^{o(\lambda)}$ for the polynomials $P_1, \ldots, P_m$.

VBPUA Assumption. Suppose we have m linear combinations of vector-input branching program evaluation polynomials, denoted as $P_1, \ldots, P_m$. To be concrete, each P_r is itself a linear combination of vector-input branching program evaluation polynomials. The only requirement is that P_r be computable by a polynomial-size arithmetic circuit. Note that such circuits may produce linear combinations of exponentially-many vector-input branching program evaluations. Suppose furthermore that the $P_1, \ldots, P_m$ polynomials are linearly independent as polynomials over the variables of the generic branching program. Assumption: There does not exist a polynomial-size circuit of degree $2^{o(\lambda)}$ that computes an annihilating polynomial for the polynomials $P_1, \ldots, P_m$.

Some clarifying details:

- The P_r polynomials can be generated using arbitrary arithmetic circuits, as long as they respect the polynomial-size restriction and represent (linear combinations of) honest evaluation polynomials. Thus, the P_r polynomials can be "adversarially" chosen to make annihilation as easy as possible.
- The requirement that the P_r polynomials be linearly independent is crucial, or else both the BPUA and the VBPUA assumptions are trivially false. Otherwise, the adversary can simply generate P_1 and $P_2 = 2P_1$, which is annihilated by $2P_1 - P_2$. We note that for linearly dependent polynomials P_r, the values t_r produced from zero-testing will satisfy the same linear relationships. As such, providing additional linearly dependent polynomials gives the adversary no additional information. Thus annihilating linearly-dependent polynomials does not give rise to an actual attack.

We note that the BPUA assumption is not stated in this language by Garg et al. [34], but that the above statement is equivalent. In particular, it is implied by the existence of PRFs in NC^1 secure against P/poly.

6.3 Evidence for the VBPUA Assumption

We currently have no way of justifying the VBPUA assumption based on simpler, well-studied assumptions. One concern is that our modifications to the regular BPUA assumption (which *can* be justified based on well-studied assumptions), despite being simple, could lead to trivially false statements. However, we believe the VBPUA assumption is far from trivially false. In Appendix E, we give some support for this belief, by demonstrating that certain simple attacks cannot break the assumption.

7 Parameter Sizes and Optimizations

In this section, we give rough, order-of-magnitude estimations for the parameter sizes of our multilinear map scheme. We focus on the setting of 4-party key exchange. Recall that our scheme does not have level zero encodings, so we use a 4-linear map instead of a 3-linear map. We also discuss some simple optimizations that greatly reduce the parameter sizes of our scheme.

Optimization 0: Pre-multiply bookends. We can shave off two levels by pre-multiplying the bookends into the left-most and right-most levels.

Optimization 1: Using Slots. CLT13 multilinear maps natively support encoding in "slots": namely, each plaintext is a vector of integers, one for each of the n primes.

So far, we have treated all slots identically, and placed the plaintext, the enforcing matrices, and an extra block diagonal in all slots. However, we can instead use the different slots for different purposes. One slot can contain the actual plaintext, one slot each can be used for the enforcing matrices, and finally one slot can be used for the extra block diagonal.

Each enforcing matrix is 2×2, and we need $\ell - 1$ enforcing matrices for each of the d meta-levels. Thus, the total number of slots we need is at most $\ell \times d$. Note that there are also $\ell \times d$, and the usual recommendation for CLT is to have the number of slots n exceed the number of levels. Therefore, we have sufficient slots. Thus, we can actually get a scheme where each meta-encoding consists of ℓ different 2×2 matrices of CLT13 encodings.

Optimization 2: Higher-order re-randomization. We need an extremely large number of encodings of zero in order to guarantee re-randomization. In order to dramatically reduce the number of encodings, we borrow the "quadratic re-randomization" trick used by Coron et al. to reduce the number of public parameter encodings in the original CLT13 scheme [18].

We switch to a degree $3R + 1$ multilinear map for some $R > 1$. The degree of the underlying CLT13 multilinear map is now $d\ell = (3R + 1)\ell$. Encodings are handed out at the three levels $\{1, \ldots, R\}, \{R + 1, \ldots, 2R\}, \{2R + 1, \ldots, 3R\}$. The user encodes their secret at level $\{3R + 1\}$. The randomizers are handed out at the singleton levels, and multiplied together to get randomizers for $\{1, \ldots, R\}, \{R+1, \ldots, 2R\}, \{2R + 1, \ldots, 3R\}$.

We note that this optimization introduces a slight heuristic component, since we have to assume that such higher-order re-randomization still fully re-randomizes. We stress however that the scheme with this optimization can still be proven secure in our model and is therefore immune to zeroizing attacks.

Estimating Parameters. An encoding in our scheme is a "meta-encoding" consisting of ℓ matrices of CLT13 encodings. Using Optimization 1, these matrices have width 2, so a meta encoding consists of 4ℓ CLT13 encodings. The underlying CLT13 map will have degree $d\ell + 2$, but we can use optimization 0 to get the degree to $d\ell$.

The overwhelming majority of the meta-encodings in the public parameters of our scheme are encodings of zero used to re-randomize at singleton levels. To estimate the number of such encodings, we count the number of encoded plaintexts as well as the amount of CLT13 randomness in each meta-encoding.

Prior to Kilian randomization, each matrix is a 2x2 diagonal, consisting of up to $2n$ entries that will need to be re-randomized (since each slot will need re-randomization). The bit-length of each plaintext in each

slot is λ. We must include an extra factor of 2, roughly, to account for the requirements of the Leftover Hash Lemma [18]. Therefore, we will need $4n\ell\lambda$ to re-randomize the plaintexts in a given level.

We must also re-randomize the CLT13 encoding randomness. There are λ bits of CLT13 encoding randomness for each of the n secret small primes. We multiply this by the number of CLT13 encodings in a meta-encoding, and include a factor of 2 to get $8n\ell\lambda$.

Adding these quantities together ensures that we have enough meta-encodings to fully re-randomize. We must provide this many meta-encodings for each of the d levels, so in total we need roughly

$$12nd\ell\lambda$$

meta-encodings in the public parameters. Using Optimization 2, we only need $(12nd\ell\lambda)^{(1/R)}$ encodings to re-randomize.

At a minimum, ℓ must be set large enough to block determinant-style attacks. Recall that with a read-once branching program, an adversary can produce a matrix of branching program outputs by varying half the input bits along the rows and half the input bits along the columns. This matrix has rank at most the width of the branching program, and thus the determinant of this matrix gives an annihilating polynomial as long as the width of this matrix is small. In a branching program that reads each input bit ℓ times, the rank of this matrix grows with w^ℓ, where w is the width. If we view our multilinear map scheme as a branching program where each input vector is read ℓ times, then we require

$$2^\ell \geq 2^\lambda$$

to block determinant-style attacks. Hence, we set $\ell = \lambda$

Following the Coron et al. suggestion, we set $n = d\ell\lambda^2$ [18]. We use the security parameter $\lambda = 80$ to compare our scheme to the obfuscation construction of Boneh et al. [39]. Before this work, the most efficient way to build secure 4-party key exchange was to go through their obfuscation scheme. We now describe several optimizations.

- Without Optimizations 1 or 2, we can re-do the calculations above, and set $\ell = 13$ and use a degree 54 CLT13 map (we can set ℓ more aggressively, since w is larger). The number of CLT13 encodings per meta-encoding is about 2^{17}, and the number of CLT13 encodings in the public parameters is about 2^{62}.
- If we introduce just optimization 2 with $R = 5$, we can set $\ell = 10$ and use a degree 160 CLT13 map. The number of CLT13 encodings per meta-encoding is about 2^{20}, and the number of CLT13 encodings in the public parameters is about 2^{33}.
- If we introduce just optimization 1, we need to set $\ell = 80$, giving a degree of 320. The number of CLT encodings per meta-encoding is 320. The number of encodings in the public parameters is about 2^{48}.
- If we introduce both optimizations 1 and 2, say with $R = 4$, the number of CLT13 encodings per meta-encoding is 320, the degree is 1040 and parameters containing 2^{19} elements. $R = 8$ gives degree 2000 and parameters containing 2^{14} elements.

For comparison, instantiating an obfuscation scheme with $\lambda = 80$ with the scheme of Boneh et al. requires 2^{44} encodings from a degree $d = 4150$ multilinear map [39]. This is without any of the overhead required to implement key exchange, which involves encrypting the program computing a PRF using a fully homomorphic encryption scheme. These findings are summarized in Section 1.1 in Table 1.

Thus, for a 4-party key exchange, our scheme substantially reduces the number of encodings and degree of the multilinear map.

While our scheme is still impractical, this is a vast improvement on the best-known techniques.

8 Discussion

This primary purpose of this work has been to identify a natural connection between zeroizing attacks on CLT13 and the problem of efficiently annihilating the set of successful zero-test polynomials, and then use it

in applications. This connection allows us to easily identify why all prior proposals to fix CLT13 by simply giving out more complicated sets of base CLT13 encodings failed. Namely, none of the candidate fixes avoided this annihilation condition.

This annihilation condition is a powerful tool for building secure constructions, as the task of annihilating a set of polynomials can be based on well-studied assumptions. Indeed, the Badrinarayanan et al. [29] obfuscator is constructed so that the successful zero-test polynomials cannot be annihilated assuming PRFs in NC^1 secure against P/poly.

The annihilation condition makes the task of directly securing CLT13 against zeroizing well-defined. Any new construction must prove that the successful zero-testing polynomials cannot be annihilated by a polynomial-size circuit. Our new map construction makes progress towards this goal, but we fall short of basing security on established assumptions.

In this field, introducing brand new assumptions to achieve secure constructions should be avoided whenever possible [47]. Unfortunately our paper is guilty of this. Nevertheless, our map is a qualitative improvement upon the state of the art. All prior CLT13 fixes designed to resist zeroizing did not derive security from any concrete assumptions beyond hope that the fixes worked. Our map derives security from an independent (though admittedly contrived) algebraic circuit complexity question about computing an annihilating polynomial.

There is much room for future work in this direction:

- Validate, refute, or simplify the Vector-Input Branching Program Un-Annihilatability Assumption. We believe there may be hope of relating our new assumption to PRFs; in particular, our assumption would be implied by a sub-exponentially secure PRF that can be computed by Vector-Input Branching Programs.
- Currently, we can only prove that successful zero-test polynomials from our multilinear maps can be decomposed into exponentially many Vector-Input Branching Program evaluations. This does not appear to be an inherent barrier, and is likely a limitation of our analysis and construction. We believe it should be possible to improve our construction (perhaps using reasonable complexity-theoretic assumptions, similar to the approach of [48]) so that each zero-test polynomial can be decomposed into polynomially many Vector-Input Branching Program evaluations. This would substantially weaken the assumption necessary for security. In particular, if this could be achieved, we would not need a sub-exponentially secure PRF computable by a Vector-Input Branching Program (a polynomially-secure PRF would then suffice).
- Identify new attacks on CLT13 that avoid our algebraic characterization of zeroizing attacks. All parameter-independent attacks on the CLT13 multilinear maps follow the zeroizing approach that we characterize in our model. Since we can now build an obfuscation scheme that provably resists such attacks, the next step in cryptanalysis is to seek a new approach to break CLT13.
- We designed our maps with the goal of achieving provable security guarantees. There is likely much room for improved efficiency.

9 Acknowledgements

We thank Amit Sahai, Tancrède Lepoint, James Bartusek for valuable discussions and feedback. We thank Leon Zhang (UC Berkeley Math) for help with Lemma 1 and the VBPUA Assumption. We also thank the anonymous reviewers for comments on prior versions of this work. Research supported in part from a DARPA SAFEWARE award and NSF. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense, the National Science Foundation, or the U.S. Government.

References

1. Boneh, D., Silverberg, A.: Applications of multilinear forms to cryptography. Contemporary Mathematics **324** (2003) 71–90
2. Garg, S., Gentry, C., Halevi, S., Sahai, A., Waters, B.: Attribute-based encryption for circuits from multilinear maps. (2013) 479–499

3. Boneh, D., Waters, B., Zhandry, M.: Low overhead broadcast encryption from multilinear maps. (2014) 206–223
4. Garg, S., Gentry, C., Sahai, A., Waters, B.: Witness encryption and its applications. (2013) 467–476
5. Garg, S., Gentry, C., Halevi, S., Raykova, M., Sahai, A., Waters, B.: Candidate indistinguishability obfuscation and functional encryption for all circuits. (2013) 40–49
6. Garg, S., Gentry, C., Halevi, S., Zhandry, M.: Functional encryption without obfuscation. (2016) 480–511
7. Sahai, A., Waters, B.: How to use indistinguishability obfuscation: deniable encryption, and more. (2014) 475–484
8. Hohenberger, S., Sahai, A., Waters, B.: Replacing a random oracle: Full domain hash from indistinguishability obfuscation. (2014) 201–220
9. Boneh, D., Zhandry, M.: Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. (2014) 480–499
10. Pandey, O., Prabhakaran, M., Sahai, A.: Obfuscation-based non-black-box simulation and four message concurrent zero knowledge for NP. (2015) 638–667
11. Chung, K.M., Lin, H., Pass, R.: Constant-round concurrent zero-knowledge from indistinguishability obfuscation. (2015) 287–307
12. Hubacek, P., Wichs, D.: On the communication complexity of secure function evaluation with long output. (2015) 163–172
13. Ananth, P.V., Sahai, A.: Functional encryption for turing machines. (2016) 125–153
14. Bitansky, N., Paneth, O., Wichs, D.: Perfect structure on the edge of chaos - trapdoor permutations from indistinguishability obfuscation. (2016) 474–502
15. Bun, M., Zhandry, M.: Order-revealing encryption and the hardness of private learning. (2016) 176–206
16. Bitansky, N., Paneth, O., Rosen, A.: On the cryptographic hardness of finding a Nash equilibrium. (2015) 1480–1498
17. Garg, S., Gentry, C., Halevi, S.: Candidate multilinear maps from ideal lattices. (2013) 1–17
18. Coron, J.S., Lepoint, T., Tibouchi, M.: Practical multilinear maps over the integers. (2013) 476–493
19. Gentry, C., Gorbunov, S., Halevi, S.: Graph-induced multilinear maps from lattices. (2015) 498–527
20. Cheon, J.H., Han, K., Lee, C., Ryu, H., Stehlé, D.: Cryptanalysis of the multilinear map over the integers. (2015) 3–12
21. Hu, Y., Jia, H.: Cryptanalysis of GGH map. (2016) 537–565
22. Coron, J.S., Lee, M.S., Lepoint, T., Tibouchi, M.: Cryptanalysis of GGH15 multilinear maps. (2016) 607–628
23. Coron, J.S., Lepoint, T., Tibouchi, M.: New multilinear maps over the integers. (2015) 267–286
24. Boneh, D., Wu, D.J., Zimmerman, J.: Immunizing multilinear maps against zeroizing attacks. Cryptology ePrint Archive, Report 2014/930 (2014) http://eprint.iacr.org/2014/930.
25. Halevi, S.: Graded encoding, variations on a scheme. Cryptology ePrint Archive, Report 2015/866 (2015) http://eprint.iacr.org/2015/866.
26. Coron, J.S., Gentry, C., Halevi, S., Lepoint, T., Maji, H.K., Miles, E., Raykova, M., Sahai, A., Tibouchi, M.: Zeroizing without low-level zeroes: New MMAP attacks and their limitations. (2015) 247–266
27. Brakerski, Z., Gentry, C., Halevi, S., Lepoint, T., Sahai, A., Tibouchi, M.: Cryptanalysis of the quadratic zero-testing of GGH. Cryptology ePrint Archive, Report 2015/845 (2015) http://eprint.iacr.org/2015/845.
28. Cheon, J.H., Fouque, P.A., Lee, C., Minaud, B., Ryu, H.: Cryptanalysis of the new CLT multilinear map over the integers. (2016) 509–536
29. Badrinarayanan, S., Miles, E., Sahai, A., Zhandry, M.: Post-zeroizing obfuscation: New mathematical tools, and the case of evasive circuits. (2016) 764–791
30. Cramer, R., Ducas, L., Peikert, C., Regev, O.: Recovering short generators of principal ideals in cyclotomic rings. (2016) 559–585
31. Albrecht, M.R., Bai, S., Ducas, L.: A subfield lattice attack on overstretched NTRU assumptions - cryptanalysis of some FHE and graded encoding schemes. (2016) 153–178
32. Cheon, J.H., Jeong, J., Lee, C.: 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.
33. Miles, E., Sahai, A., Zhandry, M.: Annihilation attacks for multilinear maps: Cryptanalysis of indistinguishability obfuscation over GGH13. (2016) 629–658
34. Garg, S., Miles, E., Mukherjee, P., Sahai, A., Srinivasan, A., Zhandry, M.: Secure obfuscation in a weak multilinear map model. (2016) 241–268
35. Lewi, K., Malozemoff, A.J., Apon, D., Carmer, B., Foltzer, A., Wagner, D., Archer, D.W., Boneh, D., Katz, J., Raykova, M.: 5Gen: A framework for prototyping applications using multilinear maps and matrix branching programs. (2016) 981–992

36. Paneth, O., Sahai, A.: On the equivalence of obfuscation and multilinear maps. Cryptology ePrint Archive, Report 2015/791 (2015) http://eprint.iacr.org/2015/791.
37. Albrecht, M.R., Farshim, P., Hofheinz, D., Larraia, E., Paterson, K.G.: Multilinear maps from obfuscation. (2016) 446–473
38. Barak, B., Garg, S., Kalai, Y.T., Paneth, O., Sahai, A.: Protecting obfuscation against algebraic attacks. (2014) 221–238
39. Boneh, D., Ishai, Y., Sahai, A., Wu, D.J.: Lattice-based SNARGs and their application to more efficient obfuscation. (2017) 247–277
40. Kilian, J.: Founding cryptography on oblivious transfer. (1988) 20–31
41. Brakerski, Z., Rothblum, G.N.: Virtual black-box obfuscation for all circuits via generic graded encoding. (2014) 1–25
42. Coron, J.S., Lee, M.S., Lepoint, T., Tibouchi, M.: Zeroizing attacks on indistinguishability obfuscation over CLT13. Cryptology ePrint Archive, Report 2016/1011 (2016) http://eprint.iacr.org/2016/1011.
43. Boneh, D., Lewi, K., Raykova, M., Sahai, A., Zhandry, M., Zimmerman, J.: Semantically secure order-revealing encryption: Multi-input functional encryption without obfuscation. (2015) 563–594
44. Coron, J.S., Lee, M.S., Lepoint, T., Tibouchi, M.: Zeroizing attacks on indistinguishability obfuscation over CLT13. (2017) 41–58
45. Agrawal, S., Gentry, C., Halevi, S., Sahai, A.: Discrete Gaussian leftover hash lemma over infinite domains. (2013) 97–116
46. Brunat, J.M., Montes, A.: A polynomial generalization of the power-compositions determinant. Linear and Multilinear Algebra **55**(4) (2007) 303–313
47. Goldwasser, S., Kalai, Y.T.: Cryptographic assumptions: A position paper. (2016) 505–522
48. Miles, E., Sahai, A., Weiss, M.: Protecting obfuscation against arithmetic attacks. Cryptology ePrint Archive, Report 2014/878 (2014) http://eprint.iacr.org/2014/878.

Appendix

A Matrix Branching Programs

In the basic setting, a single-input matrix branching program consists of a sequence of pairs of matrices, fixed left and right 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 to give a scalar. 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 a 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. [29], 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 5. *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 input selection function of BP. $BP : \{0,1\}^n \to \mathbb{Z}$ is computed as

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

B Coron et al. General Attack

We consider the general attack strategy of [26] 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 [44], but explain how each step of their attack works in our model.

Coron et al. [26] 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 [26], 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

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

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 $[B_0]_{p_j} \cdot [B_1]_{p_j}^{-1}$. 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

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

which we can rewrite as

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

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 $\{\widehat{s}_{j,i}\}_j$.

C A New Lemma

In this section, we state a generalization of a lemma by Badrinarayanan et al. [29]. Consider the same collection of matrices as in Section 2.4. 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 6. *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 7. *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.

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 [29]; 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 [29]). 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.

D 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 [34] 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 $\Pr[X_i = x_i] > 0$. 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 3. *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 [34]. We note that the original statement of this lemma in [34] uses a finite field instead of $\mathbb{Q}$, but the proof easily extends to the infinite case.

E Justification for the Vector-Input Branching Program Un-Annihilatability Assumption

In this section, we give some evidence for the Vector-Input Branching Program Un-Annihilatability (VBPUA) Assumption. As we have been unable to justify the full assumption, most of our evidence is informal and speculative.

Additionally, in this section we will only consider vector-input branching program evaluations that are integer combination products. In other words, every input vector is a k-dimensional vector of non-negative integers summing to ℓ. We denote this set as $H_{\ell,k}$.

First, we consider the assumption in the case where the inputs are restricted to a single vector.

Our restriction that the $\{P_r\}_r$ polynomials be linearly independent (over the formal variables of the generic VBP evaluations) is equivalent to the following condition. Write each $P_r = \sum_{x \in H_{\ell,k}} \alpha_{r,x} f(x)$ and verify that the $\{\alpha_r\}_r$ are linearly independent (interpreted as vectors over all x). This follows since the VBP evaluations $f(x)$ restricted to $x \in H_{\ell,k}$ are linearly independent (as these are precisely "integer combination products", whose linear independence was established in Lemma 1).

When considering matrices in a single-vector VBP, we imagine them arranged into ℓ columns of k matrices each. The same input vector is read to take a linear combination of the matrices in each column, and the resulting ℓ matrices are multiplied together. For this analysis, we ignore the bookend vectors, as we can assume they are pre-multiplied into the first and last columns. $\mathbf{B}_{i,j}$ will denote the jth matrix in the ith column.

Claim. Given a set of $\{P_r\}_r$ where $P_r = \sum_{x \in H_{\ell,k}} \alpha_{r,x} f(x)$ and $\{\alpha_r\}_r$ are linearly independent (interpreted as vectors over all x), there do not exist any linear polynomials Q such that $Q(\{P_r\}_r) \equiv 0$ as a polynomial over the formal variables of the generic VBP.

Proof. We can actually prove a significantly stronger claim. We show that all $|H_{\ell,k}| = \binom{\ell+k-1}{\ell}$ outputs of single-vector generic VBPs are linearly independent as polynomials, even when the matrices are treated as commutative scalar variables. If the output polynomials are linearly independent in this setting, they remain linearly independent when we replace these scalars with $w \times w$ matrices of formal variables.

We further assume that the set of matrices in each column is the same (again, linear independence in this restrictive setting implies linear independence in the general setting). Thus, for all i, we replace the matrix $\mathbf{B}_{i,j}$ with the scalar variable B_j. Under these simplifying assumptions, the set of generic VBP evaluation polynomials for all inputs $x \in H_{\ell,k}$ is

$$\{(\sum_{j=1}^{k} x_j B_j)^\ell \mid x_j \in \mathbb{Z}_{\geq 0} \ \forall j, \sum_{j=1}^{k} x_j = \ell\}.$$

Note that in these polynomials, B_j's are variables and coefficients are products of the x_j's. Brunat and Montes show that these polynomials are linearly independent for any positive integers ℓ, k [46]. Essentially, they give a formula for the determinant of the matrix obtained by expanding these polynomials in terms of the degree ℓ monomials (note that the number of degree ℓ monomials in k variables is also $|H_{\ell,k}|$, so the matrix is square). For any positive integers ℓ, k, this matrix has non-zero determinant.

Conjecture 1. Given a set $\{P_r\}_r$ where $P_r = \sum_{x \in H_{\ell,k}} \alpha_{r,x} f(x)$ and $\{\alpha_r\}_r$ are linearly independent (interpreted as vectors over all x), there do not exist any degree $d \leq \ell$ polynomials Q such that $Q(\{P_r\}_r) \equiv 0$ as a polynomial over the formal variables of the generic VBP.

We believe the previous claim can be extended up to degree $d \leq \ell$ polynomials, but we have been unable to prove it. Consider again the setting where all matrices are treated as commutative scalars, but now without the assumption that each column of matrices in the VBP are the same. Thus, the set of evaluations of a single-vector VBP is

$$\{\prod_{i=1}^{\ell} \sum_{j=1}^{k} x_j B_{i,j} \mid x_j \in \mathbb{Z}_{\geq 0} \ \forall j, \sum_{j=1}^{k} x_j = \ell\}.$$

For any $x \in H_{\ell,k}$, let $f(x) = \prod_{i=1}^{\ell} \sum_{j=1}^{k} x_j B_{i,j}$. Note that $f(x)$ takes a vector of integers x and outputs a polynomial in the $B_{i,j}$ variables. A degree d polynomial that annihilates the generic VBP evaluation polynomials $\{f(x)\}_{x \in H_{\ell,k}}$ is a homogeneous polynomial without loss of generality, since every $f(x)$ is homogeneous in the $B_{i,j}$ variables.

Thus, we can view a degree d annihilating polynomial as a linear combination over all possible products of d such $f(x)$ polynomials. Furthermore, it suffices to consider the case where $d = \ell$, since this implies linear independence when $d < \ell$.

Since we have $|H_{\ell,k}| = \binom{\ell+k-1}{\ell}$ distinct $f(x)$ polynomials, the number of ways to pick distinct products of d such polynomials is

$$\binom{\binom{\ell+k-1}{\ell}+d-1}{d}.$$

This expression comes from the fact that the products are in one-to-one correspondence with cardinality d multi-sets over $\binom{\ell+k-1}{\ell}$ elements.

We have reason to believe these polynomials are linearly independent when $d = \ell$. First, we note these polynomials are in a space of dimension at most

$$\binom{\binom{d+k-1}{d}+\ell-1}{\ell}.$$

To see why, consider the following procedure to pick the degrees in a monomial. For any fixed i, there are $\binom{d+k-1}{d}$ ways to assign a total degree of d to the $B_{i,j}$, and a choice between each of these $\binom{d+k-1}{d}$ assignments is made for ℓ different values of i. We say that two monomials generated using the same choices in the above procedure have the same "degree profile", and it is not hard to see that any polynomial that is a product of d $f(x)$ polynomials will have the same coefficient on all monomials with the same degree profile. The number of degree profiles is $\binom{\binom{d+k-1}{d}+\ell-1}{\ell}$, which equals the number of polynomials when $d = \ell$. For $d < \ell$ the number of polynomials is bounded above by the number of degree profiles, and for $d > \ell$ the number of polynomials is bounded below by the number of degree profiles.

Thus, we can write a matrix of coefficients of these polynomials restricted to monomials with distinct degree profiles. This matrix is square with width $\binom{\binom{\ell+k-1}{\ell}+\ell-1}{\ell}$. If this matrix is full-rank for all ℓ, k, this would imply Conjecture 1. Unfortunately, we have been unable to prove this. We have verified this conjecture for small values of ℓ, k, d. For example, when $\ell = k = d = 3$, this is indeed a full rank 220×220 matrix.

We note this analysis shows that there exists an exponential-complexity annihilating polynomial of degree $d = \ell+1$, since the number of polynomials exceeds the dimension of the space. However, this is still only for the case where the matrices are treated as commutative scalars. It seems unlikely that an annihilating polynomial derived from the scalar case will still annihilate when the $B_{i,j}$ variables are replaced by matrices. Note that exponential-complexity is insufficient to refute the VBPUA assumption, which requires a polynomial-complexity annihilating polynomial.

To justify the full VBPUA assumption, we must extend this analysis to the setting where VBPs can take n different input vectors, not just one.

Claim. Given a set $\{P_r\}_r$ where $P_r = \sum_{x \in H^n_{\ell,k}} \alpha_{r,x} f(x)$ and $\{\alpha_r\}_r$ are linearly independent (interpreted as vectors over all x), there do not exist any linear polynomials Q such that $Q(\{P_r\}_r) \equiv 0$ as a polynomial over the formal variables of the generic VBP.

This is almost identical to the claim for the single-vector case, except x is now a sequence of n vectors from $H_{\ell,k}$.

Proof. This follows from the linear independence of generic single-vector VBP evaluation polynomials. If we again treat the matrices in this VBP as scalars, we can use commutativity to rewrite $f(x)$ as a product of n generic single-vector VBP evaluation polynomials, each over a distinct set of variables. If there exists a linear combination of these products that evaluates to 0, at least one of these n generic single-vector VBP's has a linearly dependent set of outputs, which is a contradiction.

Reasoning about scalars, however, will not be sufficient to extend any claims about un-annihilatability of single-vector VBPs to the general case. Intuitively, this is due to the fact that determinant-style attacks are possible with multiple vector inputs. Consider a matrix where one vector input indexes the rows and another indexes the columns (and any remaining vectors inputs are fixed), and the entries are the corresponding generic VBP outputs. The rank of this matrix grows with w^ℓ where w is the width of the matrices in the VBP. In the scalar case $w = 1$, so the determinant is an annihilating polynomial.

We think there may be hope to base the general VBPUA assumption on un-annihilatability of single-vector VBPs and the original branching program un-annihilatability (BPUA) assumption of Garg et al. [34], but this is pure speculation. Note that we show that each VBP evaluation can be written as a linear combination of integer combination products. A VBP over integer combination products can be inefficiently transformed into a matrix branching program where each input digit is over an $|H_{\ell,k}|$-sized alphabet. This is done by simply pre-computing the matrices that arise from each vector-input, and associating the resulting matrices with the input digits. With this transformation, we can interpret a polynomial that annihilates a general VBP as a polynomial that annihilates this highly structured matrix branching program. Roughly, if the polynomial annihilates without taking advantage of the structure of the matrices, then it should violate the ordinary BPUA assumption. If the polynomial uses the structure of the matrices to annihilate, it should annihilate single-vector VBPs, as the single-vector case intuitively captures all the dependencies between the matrices. Unfortunately, we have not been able to turn this intuition into a proof.