

Return of GGH15: Provable Security Against Zeroizing Attacks

James Bartusek*, Jiaxin Guan**, Fermi Ma***, and Mark Zhandry†

Princeton University

Abstract. The GGH15 multilinear maps have served as the foundation for a number of cutting-edge cryptographic proposals. Unfortunately, many schemes built on GGH15 have been explicitly broken by so-called “zeroizing attacks,” which exploit leakage from honest zero-test queries. The precise settings in which zeroizing attacks are possible have remained unclear. Most notably, none of the current indistinguishability obfuscation (iO) candidates from GGH15 have any formal security guarantees against zeroizing attacks.

In this work, we demonstrate that all known zeroizing attacks on GGH15 implicitly construct *algebraic relations* between the results of zero-testing and the encoded plaintext elements. We then propose a “GGH15 zeroizing model” as a new general framework which greatly generalizes known attacks.

Our second contribution is to describe a new GGH15 variant, which we formally analyze in our GGH15 zeroizing model. We then construct a new iO candidate using our multilinear map, which we prove secure in the GGH15 zeroizing model. This implies resistance to all known zeroizing strategies. The proof relies on the Branching Program Un-Annilatability (BPUA) Assumption of Garg et al. [TCC 16-B] (which is implied by PRFs in NC¹ secure against P/poly) and the complexity-theoretic p -Bounded Speedup Hypothesis of Miles et al. [ePrint 14] (a strengthening of the Exponential Time Hypothesis).

1 Introduction

1.1 Motivation

Multilinear maps [1] are a powerful cryptographic tool that have enabled many cryptographic applications, ranging from multiparty key agreement [1] to extremely powerful indistinguishability obfuscation (iO) [2]. There are currently three families of multilinear maps: those of Garg, Gentry, and Halevi [3] (GGH13), those of Coron, Lepoint, and Tibouchi [4] (CLT13), and those of Gentry, Gorbunov, and Halevi [5] (GGH15).

Each of these multilinear map families are based on fully homomorphic encryption (FHE) schemes. However, the FHE schemes are intentionally weakened by providing a broken secret key to allow useful information to be extracted from encrypted values. Because of these broken secret keys, extensive cryptanalysis is required before we can gain confidence that some security remains. In this work, we study the GGH15 multilinear maps. We believe these maps are particularly interesting for a couple reasons:

- In some cases, by specializing the GGH15 construction to certain settings, security can actually be proved based on the well-studied Learning with Errors (LWE) assumption [6]. Notably, the lockable obfuscation constructions of Wichs and Zeldelis [7], of Goyal, Koppula, and Waters [8], and of Chen, Vaikuntanathan, and Wee [9], and the private puncturable PRFs of Canetti and Chen [10] and Chen et. al. [9], are all based in part on the GGH15 multilinear maps, and can be proved secure under LWE.¹ Therefore, the GGH15 multilinear maps seem to be the most promising route to achieving security based on LWE.

* bartusek.james@gmail.com

** jiaxin@guan.io

*** fermima1@gmail.com

† mzhandry@princeton.edu

¹ The lockable obfuscation constructions in [7] and [8] use ideas from prior work of Goyal, Koppula, and Waters [11] which introduced techniques for using GGH15 encodings to encrypt branching programs.

- The other two candidate multilinear maps, GGH13 and CLT13, have been shown vulnerable to quantum attacks [12–16]. In contrast, given the positive results above and the fact that LWE appears resistant to quantum attacks, it seems reasonable to expect that GGH15 is quantum immune, at least in certain settings. This leaves GGH15 as the main candidate multilinear map for the post-quantum era.

Despite the above positive results, there is still a large gap between what is provably secure under LWE and what the community hopes to achieve with multilinear maps, namely iO. On the positive side, “direct attacks” on the multilinear maps seem unlikely. Here “direct attacks” refer to attempts to attack the underlying FHE schemes, ignoring the extra information provided through the broken secret key.

Unfortunately, all multilinear map candidates have been subject to very strong “zeroizing” attacks [3, 17, 18] which exploit the broken secret key. These attacks have broken many of the applications which had not been proven secure. Since the original attacks, the field has seen a continual cycle of breaking schemes and fixing them. In the case of GGH15, these attacks [18, 19, 9] have broken many applications, including multiparty key agreement, and several of the iO candidates.

Given the importance of iO, it is important to study the security of multilinear maps even in the setting that lacks a security proof under well-studied assumptions. In order to break free from the cycle above, our aim is to develop a rigorous and formal justification for security, despite the lack of “provable” security.

Recent works have shown how to break the attack-fix-repeat cycle for GGH13 [20] and CLT13 [21] multilinear maps by devising abstract “zeroizing” models that capture and generalize all known zeroizing attack strategies on the maps. These works formally prove security of applications in these models, demonstrating in a rigorous sense that the analyzed schemes are resistant to known zeroizing attacks. Since these works, all subsequent classical polynomial-time attacks have fit the proposed models, demonstrating that these models may reasonably reflect the security of the maps.

Our goal is to extend these works to the GGH15 setting, devising a model that captures and generalizes all known zeroizing attack strategies. For GGH15, however, there are unique challenges that make this task non-trivial:

- The underlying mathematics of the scheme differs from previous schemes, and the details of the attacks are quite different. As such, any attack model will be different.
- There does not appear to be a single unified GGH15 multilinear map in the literature, but instead many variants — the basic GGH15 map, a version with safeguards, a version with commutative plaintexts, etc. Moreover, many applications do not conform to the multilinear map interface, and are instead described directly on the GGH15 implementation. The many variants of GGH15 and applications are accompanied by similarly varied settings for the attacks.
- Additionally, there are some functional limitations of GGH15: plaintexts are required to be “short”, by default plaintexts do not commute, and the level structure derives from graphs instead of sets. These present challenges in applying the standard multilinear map tools (such as Kilian randomizing branching programs, straddling sets, etc) to the GGH15 setting. This breaks many of the analysis techniques that have been applied to other multilinear map candidates, and has also led to some ad hoc proposals, such as using diagonal matrices for the plaintexts, multiplying by random scalars to create levels, or Kilian randomizing using special types of matrices.

Therefore, our goal will be to:

*Develop an abstract zeroizing attack model that captures
all known zeroizing attacks on all variants of GGH15, and
develop new techniques for proving security in this model.*

Our Results. In this work we devise an abstract attack model that applies to all existing variants of GGH15 and applications built on top of GGH15. We demonstrate that our attack model captures and generalizes all zeroizing attacks.

We then describe a new variant of GGH15, based on several prior works in the area, which we can prove strong security statements about in our model. Our new scheme is flexible enough to support a simple

obfuscation scheme which we can prove secure in our model. The result is a scheme that is provably resistant to algebraic zeroizing attacks. Before giving our results, we start with a very brief overview of the GGH15 maps and known attacks

1.2 The GGH15 Multilinear Map

GGH15 is a “graph-induced” multilinear map, which departs somewhat from the usual multilinear map notions. Here, we have a connected directed acyclic graph $G = (V, E)$ of d nodes with a single source (labeled 1) and a single sink (labeled d). A “level” is a pair of vertices (u, v) for which there is a path from u to v ; we will denote such levels by $u \rightsquigarrow v$ (different paths between u, v will be considered the same level). Plaintexts \mathbf{S} are encoded relative to levels $u \rightsquigarrow v$, and we denote such an encoding as $[\mathbf{S}]_{u \rightsquigarrow v}$.

Given a handful of encodings, the following operations can be performed:

- **Addition:** Two encodings $[\mathbf{S}_0]_{u \rightsquigarrow v}, [\mathbf{S}_1]_{u \rightsquigarrow v}$ relative to the same pair of vertices can be added, obtaining the encoding of the sum $[\mathbf{S}_0 + \mathbf{S}_1]_{u \rightsquigarrow v}$ (relative to the same pair of vertices).
- **Multiplication:** Two encodings $[\mathbf{S}_0]_{u \rightsquigarrow v}, [\mathbf{S}_1]_{v \rightsquigarrow w}$ whose nodes form a path $u \rightsquigarrow v \rightsquigarrow w$ can be multiplied, obtaining an encoding $[\mathbf{S}_0 \cdot \mathbf{S}_1]_{u \rightsquigarrow w}$ of the product at the level corresponding to concatenating the paths.
- **Zero Testing:** Given an encoding $[\mathbf{S}]_{1 \rightsquigarrow d}$ between the unique source and sink, we can test whether or not \mathbf{S} is equal to 0.

In GGH15, the “plaintexts” are also matrices, rather than scalars, meaning the multiplications above are non-commutative. Moreover, in GGH15, the plaintext matrices are required to be “short”.

GGH15 works as follows. Associated to each node u is a matrix \mathbf{A}_u . An encoding of \mathbf{S} at level $u \rightsquigarrow v$ is a matrix \mathbf{D} that satisfies $\mathbf{A}_u \mathbf{D} = \mathbf{S} \mathbf{A}_v + \mathbf{E} \pmod{q}$ where both \mathbf{D} and \mathbf{E} are “short”. This encoding is generated using a lattice trapdoor.

Addition is straightforward to verify. For multiplication, suppose $\mathbf{A}_u \mathbf{D}_0 = \mathbf{S}_0 \mathbf{A}_v + \mathbf{E}_0 \pmod{q}$ and $\mathbf{A}_v \mathbf{D}_1 = \mathbf{S}_1 \mathbf{A}_w + \mathbf{E}_1 \pmod{q}$. Then $\mathbf{A}_u \mathbf{D}_0 \mathbf{D}_1 = \mathbf{S}_0 \mathbf{S}_1 \mathbf{A}_w + \mathbf{E}_0 \mathbf{D}_1 + \mathbf{S}_0 \mathbf{E}_1 \pmod{q}$.

Since $\mathbf{S}_b, \mathbf{D}_b$ and \mathbf{E}_b are short, we can define $\mathbf{E}_2 = \mathbf{E}_0 \mathbf{D}_1 + \mathbf{S}_0 \mathbf{E}_1$, which is also short, and we see that $\mathbf{D}_0 \mathbf{D}_1$ is an encoding of $\mathbf{S}_0 \mathbf{S}_1$ relative to the path $u \rightsquigarrow w$.

For zero-testing, we note that if we have an encoding \mathbf{D} of \mathbf{S} relative to $1 \rightsquigarrow d$ and we compute $\mathbf{A}_1 \mathbf{D} \pmod{q} = \mathbf{S} \mathbf{A}_d + \mathbf{E} \pmod{q}$, the resulting matrix will be “short” relative to q if $\mathbf{S} = 0$, and otherwise, we would expect the result to be large relative to q .

1.3 Zeroizing Attacks on GGH15

As with all current multilinear map candidates, GGH15 is vulnerable to “zeroizing” attacks. These attacks leverage the fact that any time a zero-test actually detects 0, the procedure also produces an equation that holds over the integers.

For GGH15, notice that zero-testing computes $\mathbf{A}_1 \mathbf{D} \pmod{q} = \mathbf{S} \mathbf{A}_d + \mathbf{E} \pmod{q}$. If $\mathbf{S} = 0$, the result is just $\mathbf{E} \pmod{q}$, which equals \mathbf{E} since \mathbf{E} is guaranteed to be short relative to q . But recall from the GGH15 description that if \mathbf{D} is the result of several multilinear map operations, \mathbf{E} depends on not just the error terms of the original encodings, but also on the plaintext values \mathbf{S} . Therefore, any successful zero-test will give an equation depending on the original plaintext values, and this equation holds over the integers. These equations can then potentially be manipulated to learn non-trivial information about the underlying plaintexts. This is the heart of all known zeroizing attacks on GGH15.

More abstractly, suppose that c plaintext matrices $\mathbf{S}_1, \dots, \mathbf{S}_c$ are encoded relative to various edges, producing the corresponding encoding matrices $\mathbf{D}_1, \dots, \mathbf{D}_c$. In all known zeroizing attacks, the adversary adds and multiplies the matrices $\{\mathbf{D}_i\}_i$ honestly (respecting the edge-constraints of the graph) to produce top-level encodings of zero.² Let $p_u(\{\mathbf{D}_i\}_i)$ denote the u -th top-level encoding of zero the adversary constructs.

² Technically, the Coron et al. attack on key exchange does not compute top-level encodings of zero, but encodings of the same matrix relative to different source-to-sink paths [18]. However, by connecting a master source node to

Each top-level zero $p_u(\{\mathbf{D}_i\}_i)$ is then zero-tested by multiplying on the left by \mathbf{A}_1 , successfully obtaining a low-norm matrix of *zero-test results*, which we denote as T_u (in some constructions, T_u is simply a scalar). The current attacks all build a new matrix \mathbf{W} whose entries are plucked from the various T_u matrices (or T_u itself in the case of a scalar). From this point, the known attacks differ in strategy from each other. But at a high level, all of them extract some piece of information from \mathbf{W} , such as its kernel or its rank, and use this information to recover non-trivial information about the hidden plaintext matrices $\{\mathbf{S}_i\}_i$.

1.4 Our Zeroizing Model for GGH15

We make the following observation: all known attacks that recover information about the plaintexts $\{\mathbf{S}_i\}_i$ from the $\{T_u\}_u$ set up an *algebraic relation* between the two (we will often refer to this relation as a polynomial). More precisely, this means that implicit in all successful zeroizing attacks on GGH15, there is a non-trivial bounded-degree polynomial Q such that

$$Q(\{T_u\}_u, \{S_{i,j,k}\}_{i,j,k}) = 0$$

holds over the integers, where $S_{i,j,k}$ denotes the (j, k) -th entry of matrix \mathbf{S}_i . In known attacks, this Q depends on the matrix \mathbf{W} in some way; however, anticipating potential new avenues for attack, we consider a much more general attack format which assumes as little as possible about the structure of the attacks. Hence, our general condition makes no reference to a matrix \mathbf{W} .

While this condition seems simple, it is not a priori obvious that any of the GGH15 zeroizing attacks actually produce such a Q . In theory, an adversary might recover information about the plaintext matrix entries $\{S_{i,j,k}\}_{i,j,k}$ through *any* efficient algorithm taking $\{T_u\}_u$ as input. We certainly cannot hope to re-express any poly-time algorithm as a polynomial over its inputs and outputs. However, we are able to show that all known attacks can be recast as procedures that uncover a Q polynomial.

Example: The CLLT16 Attack. In Coron et al. [18], the first step of the attack is to construct the matrix \mathbf{W} as above, and then compute a vector \mathbf{v} in the left kernel of \mathbf{W} . They show, using the algebraic structure of GGH15, that such a \mathbf{v} in fact gives a relation amongst the plaintext elements only (no error terms). In particular, there is a vector \mathbf{x} of fixed polynomials in the underlying plaintext elements such that \mathbf{v} is orthogonal to \mathbf{x} . The attack then proceeds to use this relation amongst the plaintexts to break the scheme.

We observe that an equivalent view of their analysis is that \mathbf{x} is in the column span of \mathbf{W} . This means that if we append the column vector \mathbf{x} to \mathbf{W} , the rank will be unchanged. Suppose for the moment that \mathbf{W} itself is full rank, and that it is one column shy of being square. Then we can capture the fact that the rank does not increase with a simple algebraic relation: the determinant of $[\mathbf{W} | \mathbf{x}]$ equals 0. Therefore, in this restricted setting where \mathbf{W} is full rank and almost square, we see that the CLLT16 attack implicitly contains a polynomial Q as desired.

In the actual attack, \mathbf{W} may not be full rank, meaning the determinant may trivially be 0 no matter what \mathbf{x} is; this means Q does not give us a useful relation over the plaintexts. Moreover, $[\mathbf{W} | \mathbf{x}]$ may not be square, so the determinant may not be defined. With a bit more effort, we can see that a polynomial Q is nonetheless implicit in the attack for general \mathbf{W} . Basically, if we knew the rank r of \mathbf{W} , we could choose a “random” matrix \mathbf{R} with $r+1$ rows, and a “random” matrix \mathbf{S} with $r+1$ columns. If we compute $\mathbf{R} \cdot [\mathbf{W} | \mathbf{x}] \cdot \mathbf{S}$, we will obtain an $(r+1) \times (r+1)$ matrix whose rank is (with high probability) identical to the rank of $[\mathbf{W} | \mathbf{x}]$. Now we can take the determinant of $\mathbf{R} \cdot [\mathbf{W} | \mathbf{x}] \cdot \mathbf{S}$ to be our algebraic relation. In practice, we do not know r , but we can guess it correctly with non-negligible probability since r is polynomially bounded.

The GGH15 Zeroizing Model. With our observations above in hand, we can define a new zeroizing model for GGH15. Roughly, the model allows the attacker to perform multilinear map operations as explicitly allowed by the multilinear map interface (i.e. following edge constraints). Then, after performing a zero-test, if the encoding actually contained a zero, the adversary obtains a handle to the elements produced by zero-testing

the original source nodes, we can assume that all GGH15 graphs have a single source. In this case, the Coron et al. attack indeed computes top-level encodings of zero.

(the \mathbf{E} matrix in the discussion above, but potentially a different quantity for different GGH15 variants). Next, the adversary tries to construct an algebraic relation Q between the zero-test results and the original plaintexts. The only restrictions we place on Q are that it must be computable by an efficient algebraic circuit, and that it must have degree that is not too large (e.g. sub-exponential). These restrictions are very conservative, as the known attacks are quite low degree and very efficiently computable.

We also discuss (Appendix A) how to relax the model even further. We consider two different relaxations. In the first one, we allow the adversary to actually manipulate the entries of the encodings \mathbf{D} individually, rather than treating the matrices monolithically. The adversary is still required to obey the edge restrictions, so for example he can only add two entries if they belong to encodings on the same path, and can only multiply two entries if they belong to encodings on joined paths. In the second relaxation, we allow the adversary to zero-test arbitrary (degree-bounded) polynomials over the encodings, which may not obey edge restrictions. However, here, we require that the adversary treats the matrices monolithically.

1.5 A New GGH15 Variant

For our next result, we describe a new GGH15 variant. Our goal with this variant is to add safeguards — some of which have been proposed in the literature — in a rigorous way that allows us to formally analyze the effectiveness of these safeguards. Our modifications to GGH15 are as follows:

Tensored Plaintexts. First, we will modify plaintexts as suggested in Chen et al. [9]. Plaintexts will still be matrices \mathbf{M} . However, before encoding, we will manipulate \mathbf{M} as follows. First, we will tensor \mathbf{M} with a random matrix \mathbf{S} . Then we will also append \mathbf{S} as a block diagonal, obtaining the matrix

$$\mathbf{S}' = \begin{bmatrix} \mathbf{M} \otimes \mathbf{S} \\ \mathbf{S} \end{bmatrix}$$

Then we will encode \mathbf{S}' as in plain GGH15. By performing this encoding, we can use the Chen et al. [9] proof to show that direct attacks (those that do not use the broken secret key) are provably impossible, assuming LWE.

Block Diagonal Ciphertexts. Next, after obtaining a plain GGH15 encoding \mathbf{D}' of \mathbf{S}' , we append a block diagonal \mathbf{B} . Each matrix \mathbf{B} will have “smallish” entries, and will be chosen independently for each encoding. These matrices will multiply independently of the encodings \mathbf{D}' . After multiplying to the top level, we will introduce bookend vectors which will combine the products of the \mathbf{D}' and \mathbf{B} matrices together. Since the \mathbf{B} matrices are small, this will not affect zero-testing.

These block diagonals are used to inject sufficient entropy into the encodings, which will be crucial for several parts of our analysis. In particular, these block diagonals will be used to prove that any attack in our zeroizing model will also lead to an attack in a much simpler “GGH15 Annihilation Model”, discussed below in Section 1.6. Their role is similar to block diagonals introduced by Garg et al. [20], in the context of GGH13 multilinear maps. However, we note that their role here is somewhat different: our block diagonals are added to the ciphertexts, whereas in [20] they are added to the plaintexts before encoding.

Kilian Randomization. As described so far, the block diagonals \mathbf{B} can simply be stripped off by the adversary, and therefore do not provide any real-world security, despite offering security in our model as discussed in Section 1.6. The reason for this inconsistency is that our model assumes the adversary treats the encoding matrices monolithically, only operating on whole encoding matrices. Such an adversary cannot decompose a block diagonal matrix into its blocks.

We therefore employ the relaxation of our model discussed above, where the adversary can manipulate the individual components of an encoding independently. This model captures any adversary’s attempts to decompose a block matrix, and potentially much more. In order to maintain security even in this relaxed model, we Kilian-randomize the encodings, which is one of the suggested safeguards from the original GGH15 paper [5].

More precisely, we associate a random matrix \mathbf{R}_u with each node u . Then, when encoding on an edge $u \rightsquigarrow v$, we left-multiply the block diagonal encoding from above by \mathbf{R}_u^{-1} , and right multiply by \mathbf{R}_v . Note that the inner \mathbf{R} matrices cancel out when multiplying two compatible encodings. Moreover, we include \mathbf{R} 's in the bookend vectors to cancel out the outer matrices when zero-testing.

This randomization, intuitively, allows us to bind the matrices \mathbf{B} to \mathbf{D}' . We formally prove in our relaxed model that the adversary learns nothing extra if it attempts to manipulate the individual matrix entries; therefore, the adversary might as well just operate monolithically on whole encodings. This allows our analysis from above to go through.

Asymmetric Levels. Finally, we introduce asymmetric levels. In an asymmetric multilinear map, plaintexts are encoded relative to subsets of $\{1, \dots, \kappa\}$. Encodings relative to the same subset can be added, and encodings relative to disjoint subsets can be multiplied. Encodings relative to the “top” level $\{1, \dots, \kappa\}$ can be zero-tested.

We do not quite obtain asymmetric multilinear maps from GGH15. Instead, we add the asymmetric level structure on top of the graph structure. That is, there is still a graph on d nodes as well as a set of asymmetric levels. Any plaintext is now encoded relative to a pair $(u \rightsquigarrow v, L)$, where $u \rightsquigarrow v$ is a path in the graph and L is a subset of $\{1, \dots, \kappa\}$. Encodings can be added as long as both the graph-induced and asymmetric levels are identical, and encodings can be multiplied as long as both sets of levels are compatible. An element can be zero-tested only if it is encoded relative to the source-to-sink path $1 \rightsquigarrow d$, and the “top” asymmetric level $\{1, \dots, \kappa\}$. Asymmetric levels are useful for creating straddling sets [22] for proving the security of obfuscation.

To achieve this functionality, we use a technique suggested by Halevi [23]. Simply associate a random scalar to each asymmetric level, and divide an encoding by the corresponding subset of level scalars. We choose the level scalars so that they cancel out if and only if they are multiplied together, corresponding to a “top”-level encoding.

We note that it is possible for an adversary to combine elements that do not conform to the asymmetric level structure. For example, an adversary can multiply two encodings with the same asymmetric level. The point is that the adversary will not be able to successfully zero-test such an encoding.

However, the ability to combine illegal elements presents some difficulty for our analysis. Namely, the adversary could combine some illegal elements, and then cancel them out later at some point prior to zero-testing. Such a procedure will generate a valid zero-test, despite being composed of illegal operations. This breaks usual security proofs relying on asymmetric levels, which assume the ability to immediately reject any illegal operations. Essentially what we get then is an “arithmetic model” for the asymmetric levels, due to Miles, Sahai, and Weiss [24]. We will therefore use the techniques from their work in order to prove security in our model.

1.6 An Annihilation Model for Our Scheme

Next, we define a GGH15 Annihilation Model which is much simpler than the zeroizing model described above. This model makes it very easy to evaluate whether a set of plaintexts could possibly lead to an attack.

Up until successful zero-tests, this model is similar to the original model described above: the adversary can combine elements as long as they respect the edges in the underlying graph G . One key difference is that encodings are also associated with an asymmetric level structure. For the asymmetric level structure, we work with the arithmetic model, which allows the adversary to combine arbitrary elements, but any zero-test must be on elements which respect the asymmetric level structure (in addition to respecting the graph level structure).

After successful zero-tests, the model changes from above. Instead of trying to compute a polynomial relation Q , the adversary simply tries to compute an annihilating polynomial Q' for the set of zero-test polynomials previously submitted (where each is evaluated over matrices of formal variables). We show that any attack on our scheme in the GGH15 zeroizing model corresponds to an attack in the GGH15 annihilation model, allowing us to focus on proving the security of schemes in the simpler to reason about annihilation model.

1.7 Zeroizing-Proof Obfuscation

We now turn to constructing obfuscation secure against zeroizing attacks. With our new GGH15 construction and models in hand, the construction becomes quite simple. As with the original obfuscator of Garg et al. [2], our obfuscator works on matrix branching programs; such an obfuscator can be “bootstrapped” to a full obfuscator using now-standard techniques (e.g. using FHE as in [2]). Our obfuscator is essentially the obfuscation construction of [25], which in turn is based on [22]. We do have some simplifications, owing to the fact that our multilinear map directly works with matrices.

- We assume the branching program is given as a “dual-input” branching program, following the same restrictions as in [22].³ Any branching program can be converted into such a dual-input program using simple transformations as described in [22].
- We instantiate our multilinear map with the single path graph G whose length matches the length ℓ of the branching program. We also use the version with asymmetric level structure, using ℓ asymmetric levels.
- We directly encode the branching program matrices. Each matrix is encoded at the asymmetric level corresponding to how it would be encoded in [25]. Its graph-induced level is chosen to be consistent with evaluation order; namely, the branching program matrices in column i are encoded at the i -th edge in G .

We can then easily prove our obfuscator is secure against zeroizing attacks. The following is a sketch of the proof: in our GGH15 annihilation model, following previous analysis of [24], we can show that under the p -Bounded Speedup Hypothesis, the only successful zero-tests the adversary can construct are linear combinations of polynomially many honest branching program evaluations. But then, any annihilation attack gives an annihilating polynomial for branching programs. We then rely on a non-uniform variant of the *Branching Program Un-Annihilability Assumption* (BPUA) of [20], which conjectures that such annihilating polynomials are computationally intractable. This assumption can be proven true under the very mild assumption that PRFs secure against P/poly and computable by branching programs exist (in particular, PRFs computable by log-depth circuits suffice).⁴

1.8 Concurrent Work: A Weak Model for CLT13

Ma and Zhandry [21] propose a weak multilinear map model for the CLT13 multilinear maps [4], which they show captures all known zeroizing attacks on CLT13. They prove that an obfuscation scheme of Badrinarayanan, Miles, Sahai, and Zhandry [25] as well as an order revealing encryption construction of Boneh et al. [26] are secure against zeroizing attacks when instantiated with CLT13. They also give a polynomial-degree asymmetric multilinear map “fix” which they prove secure in their model under a new assumption they call the “Vector-Input Branching Program Un-Annihilability Assumption,” a strengthening of the BPUA Assumption.

Due to the substantial differences between the CLT13 and GGH15 multilinear maps, the techniques of Ma and Zhandry do not apply to the GGH15 setting. Most notably, their model captures an attacker’s ability to perform a step that leads to factoring the CLT13 modulus. There is no composite modulus in the GGH15 scheme and thus the zeroizing attacks we consider are quite different.

1.9 Discussion and Limitations

We emphasize that our security claims are far from implying a provably secure iO candidate. Instead, what we achieve is security against the two major vulnerabilities of GGH15-based constructions. The first

³ Dual-input is necessary to invoke the p -Bounded Speedup Hypothesis for MAX 2-SAT. This arises in the proof of Lemma 8.

⁴ Using similar arguments, we can adapt the order-revealing encryption (ORE) construction of [26] to our scheme, and prove security under BPUA, analogous to constructing ORE from GGH13 as in [20].

vulnerability, as Gentry et al. note in their original paper, is that the GGH15 construction requires high-entropy plaintexts to have any hope of security. We address this by incorporating the “generalized GGH15 encodings” of Chen et al. [9], which introduce the necessary pre-encoding entropy. The second vulnerability arises from algebraic zeroizing attacks on GGH15. Until now, no GGH15 obfuscation candidate has come with any formal proofs of zeroizing resistance. Indeed, Chen et al. (Section 1.4 of [9]) state the goal of achieving zeroizing-resistance for a GGH15 obfuscator as an important open problem. With our GGH15 zeroizing model characterization of zeroizing attacks, we are able to give a formal security guarantee against such attacks.

The obvious limitation of our approach is that new vulnerabilities in GGH15 may arise at any point in the future. Like any proof of security in an idealized model, our guarantees only hold against restricted adversaries and should not be taken as a robust claim of iO security. In the language of Goldwasser and Kalai [27], we view the security proof of our candidate as an *intermediate result* toward the goal of iO from well-studied assumptions. Nevertheless, we believe the uncertain state of current GGH15-based obfuscators makes our iO candidate both theoretically and practically relevant:

- From a theoretical point of view, the longer that algebraic zeroizing strategies remain the only way to break GGH15, the more confidence our iO candidate and GGH15 zeroizing model will inspire. On the other hand, if our candidate is broken, the cryptanalysis will need to introduce new, non-zeroizing techniques, or use radically different zeroizing strategies that cannot be expressed as algebraic relations. In other words, our proofs show that current zeroizing techniques only extend so far, and highlights the need for new ideas in GGH15 cryptanalysis.
- From a practical standpoint, candidate GGH15-based obfuscators have already been implemented. One example is the recent candidate of Halevi et al. [28] which is now known to be broken by the zeroizing attack of Chen et al [9]. In light of this, we strongly believe that future obfuscation implementations should come with a formal justification of security against zeroizing attacks. Our security arguments present a new way to achieve this, though we do not claim our specific candidate is practical.

2 Preliminaries

2.1 Notation

Throughout this paper we use capital bold letters to denote a matrix \mathbf{M} . Lowercase bold letters denote vectors \mathbf{v} . Occasionally, we will use $diag(\mathbf{M}_1, \dots, \mathbf{M}_k)$ to denote a matrix with block diagonals $\mathbf{M}_1, \dots, \mathbf{M}_k$. We will often need to distinguish between values and formal variables. For example, in a situation where the variable $x = 2$, it can be difficult to tell when x represents a formal variable or when it represents the number 2. Thus, whenever we want x to denote a formal variable, we explicitly write it as \hat{x} . When an expression over formal variables is identically 0, we write \equiv (or $\not\equiv$ if it is not). Finally, we identify the ring \mathbb{Z}_q with elements $[-q/2, q/2]$.

2.2 Background on Lattices

Here, we give a very brief background on lattices. A lattice Λ of dimension n is a discrete additive subgroup of \mathbb{R}^n that is generated by n basis vectors denoted as $\{\mathbf{b}_1, \dots, \mathbf{b}_n \in \mathbb{R}^n\}$. Specifically, we have $\Lambda = \{\sum_{i \in [n]} x_i \cdot \mathbf{b}_i\}$ for integer x_i ’s. We then have the following useful definitions and lemmas.

Definition 1 (Discrete Gaussian on Lattices). *First, define the Gaussian function on \mathbb{R}^n with center $\mathbf{c} \in \mathbb{R}^n$ and width $\sigma > 0$ as*

$$\forall \mathbf{x} \in \mathbb{R}^n, \rho_{\sigma, \mathbf{c}}(\mathbf{x}) = e^{-\pi \|\mathbf{x} - \mathbf{c}\|^2 / \sigma^2}.$$

Then, the discrete Gaussian distribution over an n -dimensional Λ with center $\mathbf{c} \in \mathbb{R}^n$ and width σ is defined as

$$\forall \mathbf{x} \in \Lambda, D_{\Lambda, \sigma, \mathbf{c}}(\mathbf{x}) = \frac{\rho_{\sigma, \mathbf{c}}(\mathbf{x})}{\sum_{\mathbf{y} \in \Lambda} \rho_{\sigma, \mathbf{c}}(\mathbf{y})}.$$

Note that we omit the subscript \mathbf{c} when it is $\mathbf{0}$.

Definition 2 (Decisional Learning with Errors (LWE) [6]). For $n, m \in \mathbb{N}$ and modulus $q \geq 2$, distributions for secret vectors, public matrices, and error vectors $\theta, \pi, \chi \subseteq \mathbb{Z}_q$, an LWE sample is defined as $(\mathbf{A}, \mathbf{s}^T \mathbf{A} + \mathbf{e}^T \pmod{q})$ with $\mathbf{s}, \mathbf{A}, \mathbf{e}$ sampled as $\mathbf{s} \leftarrow \theta^n$, $\mathbf{A} \leftarrow \pi^{m \times n}$, and $\mathbf{e} \leftarrow \chi^m$.

An algorithm is said to solve $\text{LWE}_{n, m, q, \theta, \pi, \chi}$ if it is able to distinguish the LWE sample from one that is uniformly sampled from $\pi^{m \times n} \times U(\mathbb{Z}_q^{m \times 1})$ with probability non-negligibly greater than $1/2$.

Lemma 1 (Hardness of LWE [6]). Given $n \in \mathbb{N}$, for any $m = \text{poly}(n)$, $q \leq 2^{\text{poly}(n)}$, let $\theta = \pi = U(\mathbb{Z}_q)$, $\chi = D_{\mathbb{Z}, \sigma}$ where $\sigma \geq 2\sqrt{n}$. If there exists an efficient (possible quantum) algorithm that breaks $\text{LWE}_{n, m, q, \theta, \pi, \chi}$, then there exists an efficient (possible quantum) algorithm for approximating SIVP and GAPSVP in the ℓ_2 norm, in the worst case, to within $\tilde{O}(nq/\sigma)$ factors.

Lemma 2 (LWE with Small Public Matrices [29]). Given n, m, q, σ chosen as in Lemma 1, $\text{LWE}_{n', m, q, U(\mathbb{Z}_q), D_{\mathbb{Z}, \sigma}, D_{\mathbb{Z}, \sigma}}$ is as hard as $\text{LWE}_{n, m, q, U(\mathbb{Z}_q), U(\mathbb{Z}_q), D_{\mathbb{Z}, \sigma}}$ for $n' \geq 2n \log q$.

Lemma 3 (Trapdoor Sampling [30]). There exists a PPT algorithm called $\text{TrapSam}(1^n, 1^m, q)$ that, given any integers $n \geq 1$, prime $q \geq 2$, and sufficiently large $m = O(n \log q)$, outputs (\mathbf{A}, τ) where \mathbf{A} is statistically close to uniform over $\mathbb{Z}_q^{n \times m}$, and τ is a trapdoor for \mathbf{A} . Furthermore, there is another PPT algorithm $\text{SampleD}(\mathbf{A}, \tau, \mathbf{y}, \sigma)$ that outputs a sample of vector \mathbf{d} from $D_{\mathbb{Z}^m, \sigma}$ conditioned on $\mathbf{Ad} = \mathbf{y}$. For sufficiently large $\sigma = O(\sqrt{n \log q})$, with all but negligible probability, we have

$$\{\mathbf{A}, \mathbf{d}, \mathbf{y} : \mathbf{y} \leftarrow U(\mathbb{Z}_q^n), \mathbf{d} \leftarrow \text{SampleD}(\mathbf{A}, \tau, \mathbf{y}, \sigma)\} \approx_s \{\mathbf{A}, \mathbf{d}, \mathbf{y} : \mathbf{d} \leftarrow D_{\mathbb{Z}^m, \sigma}, \mathbf{y} = \mathbf{Ad}\}.$$

2.3 GGH15 Graph-Induced Maps

The GGH15 construction is parameterized by a directed graph $G = (V, E)$ with a single source and sink, a ring R , and some integer parameters n, m, q, σ, ν with $m \gg n$. The plaintext space consists of matrices $\mathbf{S} \in R^{n \times n}$ whose entries are “short” (much smaller than q). The encodings are matrices $\mathbf{D} \in (R/qR)^{m \times m}$. Each plaintext and encoding matrix is associated with a specific edge or path in the graph. At parameter generation, a random public matrix $\mathbf{A}_u \in R^{n \times m}$ is sampled for each vertex u , along with secret trapdoor information τ_u . An encoding of a plaintext matrix $\mathbf{S} \in R^{n \times n}$ with respect to a path $(u \rightsquigarrow v)$ is a matrix \mathbf{D} that satisfies

$$\mathbf{A}_u \mathbf{D} = \mathbf{S} \mathbf{A}_v + \mathbf{E} \pmod{q}. \quad (1)$$

for a low-norm error matrix $\mathbf{E} \in R^{n \times m}$. Such a \mathbf{D} can be computed by a secret key holder with the trapdoor τ_u .

Two encodings \mathbf{D}_1 and \mathbf{D}_2 can be added as matrices provided they are encoded relative to the same path. Multiplication is allowed as long as \mathbf{D}_1 is an encoding relative to $u \rightsquigarrow v$ and \mathbf{D}_2 is encoded relative to a path that starts at v . The resulting encoding, $\mathbf{D}_1 \cdot \mathbf{D}_2$ is relative to the combined path $u \rightsquigarrow w$. The encoding invariant (Equation 1) is preserved under a limited number of additions and multiplications as long as the initial \mathbf{S} , \mathbf{E} , and \mathbf{D} matrices are sufficiently short.

The encoding scheme supports zero-testing on encodings relative to source-to-sink paths. For a given public source matrix \mathbf{A}_s and a source to sink encoding \mathbf{D} , zero-testing involves checking if the product $\mathbf{A}_s \mathbf{D}$ is sufficiently small. If \mathbf{D} is an encoding of the 0 matrix, it is straightforward to see that their product is a small matrix:

$$\mathbf{A}_s \mathbf{D} = 0 \cdot \mathbf{A}_s + \mathbf{E} = \mathbf{E} \pmod{q}.$$

2.4 Matrix Branching Programs

We introduce dual-input matrix branching programs of the type considered in [31] but with one minor modification. Formally, a dual-input matrix branching program BP of length h , width w , and input length ℓ consists of an input selection function $\text{inp} : [h] \rightarrow [\ell] \times [\ell]$ and $4h$ matrices

$$\{\mathbf{M}_{i,b_1,b_2} \in \{0, 1\}^{w \times w}\}_{i \in [h]; b_1, b_2 \in \{0, 1\}}.$$

BP is evaluated on input $x \in \{0, 1\}^\ell$ by checking whether or not

$$\prod_{i \in [h]} \mathbf{M}_{i,x(i)} = 0^{w \times w}$$

where $x(i) := (x_{\text{inp}(i)_1}, x_{\text{inp}(i)_2})$. Note that the definition from [31] includes right and left bookend vectors that are multiplied on either side of the branching program product resulting in a scalar that is either zero or non-zero. We can simply turn each bookend into a matrix by repetition of rows/columns in order to recover the functionality described above. As noted in [31], branching programs of this type can be constructed from any NC^1 circuit with $h = \text{poly}(n)$ and $w = 5$ by Barrington's theorem [32].

2.5 Straddling Sets

Our obfuscator uses the notion of straddling sets in order to enforce input consistency.

Definition 3 (Straddling Set System). A straddling set system with n entries is a universe set \mathbb{U} and a collection of subsets $\mathbb{S} = \{S_{i,b} \subseteq \mathbb{U}\}_{i \in [n], b \in \{0, 1\}}$ such that

- $\bigcup_{i \in [n]} S_{i,0} = \bigcup_{i \in [n]} S_{i,1} = \mathbb{U}$
- For any distinct $C, D \subseteq \mathbb{S}$ such that $\bigcup_{S \in C} S = \bigcup_{S \in D} S$, there exists $b \in \{0, 1\}$ such that $C = \{S_{i,b}\}_{i \in [n]}$ and $D = \{S_{i,1-b}\}_{i \in [n]}$

From [22], the following is a straddling set system with n entries over universe $\mathbb{U} = \{1, \dots, 2n - 1\}$

$$\begin{aligned} S_{1,0} &= \{1\}, S_{2,0} = \{2, 3\}, \dots, S_{i,0} = \{2i - 2, 2i - 1\}, S_{n,0} = \{2n - 2, 2n - 1\} \\ S_{1,1} &= \{1, 2\}, \dots, S_{i,1} = \{2i - 1, 2i\}, \dots, S_{n-1,1} = \{2n - 3, 2n - 2\}, S_{n,1} = \{2n - 1\} \end{aligned}$$

3 GGH15 Zeroizing Model

3.1 Graph-Induced Ideal Model

We discuss the syntax of graph-induced graded encoding schemes and describe an ideal model (also known as a generic multilinear map model) for the graph-induced setting. Note that this is completely analogous to the ideal model for symmetric/asymmetric multilinear maps, which itself is an extension of the generic group model to the multilinear map setting [3, 2].

We consider directed acyclic graphs (DAGs) $G = (V, E)$ where $|V| = d$. We assume the graph has a single source and a single sink. We label the vertices from 1 to d according to some fixed topological ordering, so that all edges/paths in the graph can be written as $j \rightsquigarrow k$ where $j, k \in [d], j < k$. (Note that the precise distinction between paths and edges in graph-induced maps is not important, since the intermediate nodes on a path do not matter).

Formally, the graph-induced ideal model is instantiated with a DAG $G = (V, E)$, a plaintext ring R , and a set of plaintexts $\{M_i, u_i \rightsquigarrow v_i\}_i$. The plaintexts are indexed by i , and plaintext M_i comes with an associated path $u_i \rightsquigarrow v_i$, where $u_i, v_i \in [d], u_i < v_i$.

We describe the model as an interaction between an oracle \mathcal{M} (the “model”) and a user \mathcal{A} (the “adversary”).

- *Instance Generation.* The model \mathcal{M} is instantiated with the graph G , plaintext ring R and the set $\{M_i, u_i \rightsquigarrow v_i\}_i$. For each i , the model \mathcal{M} generates a handle \widehat{C}_i , stores a pointer from \widehat{C}_i to M_i , and releases $(\widehat{C}_i, u_i \rightsquigarrow v_i)$ publicly.

\mathcal{A} can only interact with the handles \widehat{C}_i , which in the ideal setting leak no information about M_i . The model provides the following interfaces for \mathcal{A} :

- *Addition.* Addition on two handles $\widehat{C}_i, \widehat{C}_j$ is permitted only if their corresponding paths $u_i \rightsquigarrow v_i, u_j \rightsquigarrow v_j$ are the same. The model \mathcal{M} looks up the corresponding plaintexts M_i, M_j , and returns a newly generated handle \widehat{C}_k to the sum $M_i + M_j$, along with the path $u_i \rightsquigarrow v_i$.
- *Multiplication.* Multiplication on two handles $\widehat{C}_i, \widehat{C}_j$ is permitted only if the path $u_i \rightsquigarrow v_i$ ends where path $u_j \rightsquigarrow v_j$ begins ($v_i = u_j$). The model \mathcal{M} looks up the corresponding plaintexts M_i, M_j , and returns a newly generated handle \widehat{C}_k to the product $M_i \cdot M_j$, along with the combined path $u_i \rightsquigarrow v_j$.
- *Zero-Test.* \mathcal{A} can request a zero-test on a handle \widehat{C} . \mathcal{M} responds with “zero” if the corresponding plaintext is 0, and the corresponding path is the source-to-sink path. Otherwise, the result is “not zero.”

Implicit in this model is the assumption that the adversary cannot learn anything beyond what the interfaces explicitly allow. In particular, it can only learn the bits returned by zero-testing honestly generated source-to-sink encodings, and nothing more.

These interfaces suffice for our purposes, but we note that a full scheme (such as the GGH15 construction) usually implements extract and re-randomization capabilities (refer to [5] for definitions and constructions). Note that an explicit encoding procedure has been omitted in this description since it is handled in instance generation. This reflects the fact that in the GGH15 instantiation of graph-induced multilinear maps (GGH15), encoding is not a public procedure.

Zero-Test Circuits Observe that addition, multiplication, and zero-testing can be handled in a single interface. Here, \mathcal{A} simply submits an arithmetic circuit p that computes a polynomial over the handles $\{\widehat{C}_i\}_i$. Any handle that results in a successful zero-test in the above model can be represented as a polynomial-size circuit over $\{\widehat{C}_i\}_i$ where each arithmetic gate respects the addition and multiplication restrictions enforced by the graph structure.

However, we can relax the restriction on the arithmetic circuit so that the individual gates may not necessarily respect the graph constraints, but the resulting polynomial still computes a valid source-to-sink encoding (for example, if terms that violate graph constraints cancel out in the final evaluation). Looking ahead to our GGH15 Zeroizing Model, we will require this relaxed constraint on arithmetic circuits, which only makes the model more conservative.

3.2 GGH15 Variants

There are a number of GGH15 variants in the literature that modify the original GGH15 construction at a number of key points. We identify several points in which the various schemes differ, and establish standard notation before introducing our model.

Pre-Processing. In the original GGH15 construction [5], an encoding of a plaintext matrix \mathbf{M} at path $u \rightsquigarrow v$ is the matrix \mathbf{D} satisfying $\mathbf{A}_u \cdot \mathbf{D} = \mathbf{M} \cdot \mathbf{A}_v + \mathbf{E}$.

A number of works have proposed performing additional pre-processing to \mathbf{M} before sampling the matrix \mathbf{D} . For example, the $\gamma_{\otimes \text{diag}}$ -GGH15 encodings of Chen et al. [9] encode a plaintext matrix \mathbf{M} by first sampling a random \mathbf{P} (in the notation of [9], this is the $\mathbf{S}_{i,b}$ matrix) and constructing the matrix $\text{diag}(\mathbf{M} \otimes \mathbf{P}, \mathbf{P})$ where \otimes denotes the tensor product (Kronecker product).

Then the encoding \mathbf{D} is the matrix satisfying

$$\mathbf{A}_u \cdot \mathbf{D} = \begin{bmatrix} \mathbf{M} \otimes \mathbf{P} \\ \mathbf{P} \end{bmatrix} \cdot \mathbf{A}_v + \mathbf{E}.$$

As other GGH15 variants perform different pre-processing steps on the initial plaintext \mathbf{M} , we denote the result of pre-processing as \mathbf{S} . If there is no pre-processing step, then $\mathbf{S} = \mathbf{M}$. In the example above $\mathbf{S} = \text{diag}(\mathbf{M} \otimes \mathbf{P}, \mathbf{P})$.⁵ The encoding is then computed as $\mathbf{A}_u \cdot \mathbf{D} = \mathbf{S} \cdot \mathbf{A}_v + \mathbf{E}$.

Post-Encoding. The original GGH15 paper [5] as well as Halevi [23] discuss various steps intended to safeguard the scheme against attacks (sometimes called “GGH15 with safeguards”). These steps essentially perform operations on the matrix \mathbf{D} generated from the standard GGH15 encoding procedure to produce a “final” encoding \mathbf{C} . We will adopt this notation, and set \mathbf{C} to be the result of the overall encoding process. If there is no post-encoding step, then $\mathbf{C} = \mathbf{D}$.

Zero-Testing. In the original GGH15 construction, zero-testing a source-to-sink encoding is done by computing a matrix from the public parameters and the encodings \mathbf{C} , and testing if this matrix is small. Ideally, only the bit of information (whether or not the result is small) is useful to the adversary. Of course, the zeroizing attacks on GGH15 show that this assumption is false, and that the actual matrix resulting from the zero-test can provide useful information to the adversary [18, 19, 9]. This matrix will be referred to as the “result” of zero-testing. To avoid confusion, the 0/1 bit learned from the zero-test will be referred to as a bit rather than the result.

In certain GGH15 variants, the result of zero-testing is not a matrix. For example in “GGH15 with safeguards” [5, 23], the result of zero-testing is a scalar. We will use the letter T to generically denote the result of zero-testing (noting that T may represent a matrix depending on the scheme, even though it might not be written in bold).

GGH15 Algorithms. Unlike the graph-induced ideal model, our GGH15 Zeroizing Model is defined with respect to a specific GGH15 scheme/variant in mind. For example, in the ideal setting, a zero-test is successful if and only if the product of the plaintexts is zero. In our GGH15 Zeroizing Model, the model explicitly maintains encodings corresponding to each plaintext, and whether a zero-test is successful is determined by performing computations on the encodings and public parameters corresponding to an actual GGH15 variant.

To specify our model, we let the scheme be denoted by G . For example, G may be the original GGH15 construction [5], the “GGH15 with safeguards” [23], etc. To be a valid GGH15 scheme, we require G to have the following algorithms (in the literature, `PreProcess` is usually implicit):

- $\mathsf{G.KeyGen}(1^\lambda, G, R, \mathsf{aux})$: Takes the security parameter, a description of a graph G with source 1 and sink d , a ring R , and potential auxiliary information aux , and produces public parameters pp and secret parameters sp .
- $\mathsf{G.PreProcess}(\mathsf{sp}, \mathbf{M})$: Converts the input plaintext \mathbf{M} into a pre-encoding \mathbf{S} . For many schemes (including the original GGH15 construction), $\mathbf{S} = \mathbf{M}$.
- $\mathsf{G.Enc}(\mathsf{sp}, \mathbf{S}, u_i \rightsquigarrow v_i)$: Encodes \mathbf{S} on the path $u_i \rightsquigarrow v_i$.
- $\mathsf{G.Add}(\mathsf{pp}, \mathbf{C}_1, \mathbf{C}_2)$: Takes an encoding \mathbf{C}_1 of \mathbf{M}_1 at path $u_1 \rightsquigarrow v_1$ and an encoding \mathbf{C}_2 of \mathbf{M}_2 at path $u_2 \rightsquigarrow v_2$. If $u_1 = u_2$ and $v_1 = v_2$, this produces an encoding \mathbf{C}_3 of $\mathbf{M}_1 + \mathbf{M}_2$ at path $u_1 \rightsquigarrow v_1$.
- $\mathsf{G.Mult}(\mathsf{pp}, \mathbf{C}_1, \mathbf{C}_2)$: Takes an encoding \mathbf{C}_1 of \mathbf{M}_1 at path $u_1 \rightsquigarrow v_1$ and an encoding \mathbf{C}_2 of \mathbf{M}_2 at path $u_2 \rightsquigarrow v_2$. If $v_1 = u_2$, this produces an encoding \mathbf{C}_3 of $\mathbf{M}_1 \cdot \mathbf{M}_2$ at path $u_1 \rightsquigarrow v_2$.
- $\mathsf{G.ZeroTest}(\mathsf{pp}, \mathbf{C})$: Takes an encoding \mathbf{C} , computes a result T , and returns (T, b) . If \mathbf{C} is an encoding of 0 relative to path $1 \rightsquigarrow d$, then T is “small” and $b = 1$ (indicating successful zero-test). Otherwise, $b = 0$ with overwhelming probability.

3.3 GGH15 Zeroizing Model

Initialize Parameters. \mathcal{M} is initialized with a security parameter λ , a graph $G = (V, E)$, a ring R , potential auxiliary information aux , and a graph-induced encoding scheme G . It runs $\mathsf{G.KeyGen}(1^\lambda, G, R, \mathsf{aux})$ to generate the public and secret parameters $(\mathsf{pp}, \mathsf{sp})$, which it stores.

⁵ Essentially, \mathbf{S} is the result of the γ functions in the notation of [9]. However, the \mathbf{S} notation is more natural for our setting, especially when referring to entries of these matrices.

Initialize Elements. \mathcal{M} is given a set of initial plaintext elements $\{\mathbf{M}_i, u_i \rightsquigarrow v_i\}_i$ where each plaintext is indexed by i , and i -th plaintext \mathbf{M}_i is associated with path $u_i \rightsquigarrow v_i$. The model applies a pre-processing procedure to the plaintext (recall in the standard GGH15 construction, this procedure does nothing):

$$\mathbf{S}_i \leftarrow \text{G.PreProcess}(\mathbf{sp}, \mathbf{M}_i).$$

Then it computes the encoding \mathbf{C}_i from the pre-encoding \mathbf{S}_i :

$$\mathbf{C}_i \leftarrow \text{G.Enc}(\mathbf{sp}, \mathbf{S}_i, u_i \rightsquigarrow v_i).$$

Each tuple $(\mathbf{S}_i, \mathbf{C}_i, u_i \rightsquigarrow v_i)$ is stored in the *pre-zero-test table*. For each encoding \mathbf{C}_i , the model generates a corresponding handle \widehat{C}_i that contains no information about \mathbf{C}_i or \mathbf{S}_i . The handle is released, along with the corresponding encoding level $u_i \rightsquigarrow v_i$, and the model internally stores a mapping between the handle \widehat{C}_i and the tuple $(\mathbf{S}_i, \mathbf{C}_i, u_i \rightsquigarrow v_i)$. While the encoding \mathbf{C}_i is a matrix, the adversary is given a single handle \widehat{C}_i to the entire matrix.

Zero-Testing. The adversary generates a polynomial p (represented as a $\text{poly}(\lambda)$ -size arithmetic circuit), over the handles \widehat{C}_i and submits it to the model. Note that since the handles correspond to non-commutative encodings, p must be treated as a polynomial over non-commuting variables.

The model verifies that p computes an edge-respecting polynomial, meaning that each monomial is a product of encodings corresponding to a source-to-sink path. If p is not edge-respecting, the model returns \perp . If p is edge-respecting, the model \mathcal{M} evaluates p on the encodings \mathbf{C}_i , producing a matrix $p(\{\mathbf{C}_i\}_i)$ that corresponds to a valid source-to-sink encoding (or a linear combination of source-to-sink encodings). Finally, \mathcal{M} zero-tests $p(\{\mathbf{C}_i\}_i)$, obtaining $(T, b) \leftarrow \text{G.ZeroTest}(\mathbf{pp}, p(\{\mathbf{C}_i\}_i))$. If the zero-test is successful ($b = 1$), the model stores the value T (possibly a matrix, vector, or scalar) and generates a handle \widehat{T}_ℓ to each element of T . Otherwise, the model returns \perp .

We index the successful zero-tests by the letter u , so T_u will denote the result of the u -th successful zero-test, \widehat{T}_u will be the corresponding handles, and p_u will be the polynomial submitted for the u -th successful zero-test.⁶

Post-Zero-Test. In the post-zero-test stage, the adversary submits a polynomial Q of degree at most $2^{o(\lambda)}$ over the handles $\{\widehat{T}_u\}_u$ and pre-encoding elements $\{\widehat{S}_{i,j,k}\}_{i,j,k}$ where $\widehat{S}_{i,j,k}$ is a handle to the (j, k) -th entry of the i -th pre-encoding matrix \mathbf{S}_i . For the sake of readability, we will frequently drop the outer subscripts and denote these sets as $\{\widehat{T}_u\}$ and $\{\widehat{S}_{i,j,k}\}$. The model \mathcal{M} checks the following:

1. $Q(\{T_u\}, \{S_{i,j,k}\}) = 0$
2. $Q(\{T_u\}, \{\widehat{S}_{i,j,k}\}) \not\equiv 0$
3. $Q(\{\widehat{T}_u\}, \{S_{i,j,k}\}) \not\equiv 0$

If all three checks pass, the model returns “Win”, and otherwise it returns \perp . In Section 3.4, we explain how we derive these conditions, and in Section 3.5 we justify how these conditions capture the known attacks. We note that \mathcal{A} is free to submit as many polynomials Q as it wants as long as it remains polynomial time. If any such Q causes \mathcal{M} to return “Win” then the adversary is successful.

Note that in reality, a zeroizing attack that succeeds with non-negligible probability is indeed considered successful. Thus, we will allow the adversary to be possibly randomized, and we define a successful adversary to be one that can obtain a “Win” with non-negligible probability (over the randomness of the model and the adversary).

⁶ Although we denote each zero-test result as T_u , an adversary is not required to use T_u monolithically. For example, an adversary can extract a single entry of T_u in the case when T_u are matrices.

3.4 Deriving the Post-Zero-Test Win Condition

All known zeroizing attacks on GGH15 exclusively rely on the results of zero-tests to recover information about the hidden plaintexts [18, 19, 9]. In our model, this can be viewed as using the values $\{T_u\}$ to learn something about the values $\{S_{i,j,k}\}$. Furthermore, we claim that all attacks that do this recover information that can be expressed as an *algebraic relation* (we justify this claim in Section 3.5).

More precisely, underneath all successful zeroizing attacks on GGH15, there is a non-trivial bounded-degree polynomial Q (the algebraic relation) such that

$$Q(\{T_u\}, \{S_{i,j,k}\}) = 0$$

holds over the integers.

This corresponds to the intuition that in a zeroizing attack, the adversary can learn something about the pre-encoding entries $S_{i,j,k}$ by plugging the results of zero-testing $\{T_u\}$ into the above relation. While not every algebraic relation is solvable, we take the conservative route and model any non-trivial relation the adversary can construct as a win.

Now we formalize what it means for Q to be non-trivial. If the adversary can indeed plug in the results of zero-testing to learn something about the $S_{i,j,k}$, then the expression must not be identically zero over the $\widehat{S}_{i,j,k}$ terms (taken as formal variables), when the $\{T_u\}$ values are plugged in. Thus, we have the condition

$$Q(\{T_u\}, \{\widehat{S}_{i,j,k}\}) \not\equiv 0.$$

We also want to ensure that the zeroizing attack uncovers information about the pre-encodings beyond what the adversary can learn honestly. Note that if the adversary obtains a successful zero-test, it learns that some function of the pre-encoding entries $\widehat{S}_{i,j,k}$ evaluates to 0. As a simple example, if the adversary learns from an honest zero-test that matrix $\mathbf{S}_{i'}$ is the 0 matrix, then $\mathbf{S}_{i',j',k'} = 0$ for any choice of j', k' . The formal polynomial $Q = \widehat{S}_{i',j',k'}$ for any j', k' would then satisfy both of the above conditions. However, we should not consider this a successful zeroizing “attack,” as it does not use the zero-test results to derive information about the pre-encodings.

To ensure that what the adversary learns about the pre-encodings relies on T_u in a non-trivial way, we enforce a third condition

$$Q(\{\widehat{T}_u\}, \{S_{i,j,k}\}) \not\equiv 0.$$

Roughly, this condition states that the relation is not always satisfied regardless of what the $\{T_u\}$ values are, and thus the attack “uses” the zero-test leakage.

3.5 Algebraic Relations in Known Attacks

We now describe in detail how in all known zeroizing attacks on GGH15, we can derive an algebraic relation Q satisfying our three win conditions with non-negligible probability. For a review of the settings of the zeroizing attacks, refer to Appendix B or the original papers [18, 19, 9].

We will first review steps that are, in large part, common to all known zeroizing attacks on GGH15. In these steps, the zeroizing attacks derive a matrix \mathbf{W} where each entry comes from successful zero-test results $\{T_u\}$. The \mathbf{W} matrix in all known attacks crucially factors as $\mathbf{W} = \mathbf{XY}$, and the various attacks rely on different properties of this factorization. We will ignore the precise details of how the attacks proceed from this point, and instead show that given this \mathbf{W} we can derive the algebraic relation Q corresponding to our win condition.

We stress that a zeroizing attack does not necessarily need to follow these steps to be captured by our win condition. We only use this template to demonstrate how the attacks fit in our model as it provides the simplest exposition.

Step 1: Compute Top-Level Encodings of Zero. All known attacks begin by computing $J \cdot K$ different top-level encodings of zero, for some positive integers J, K . This corresponds to the adversary submitting $J \cdot K$ edge-respecting polynomials p in the zero-testing stage of our model.

Specifically, for each $j \in [J], k \in [K]$, the attacks compute a particular polynomial $\{p_{j,k}\}_{j \in [J], k \in [K]}$ over the encodings $\{\mathbf{C}\}$.

To specify the specific polynomials over the encodings in each of the various attacks, we deviate slightly from our generic notation in Section 3.3. In our generic notation we denote the u -th zero-test polynomial as p_u , but here we will write the zero-test polynomials as $p_{j,k}$.

- The CLLT16 attack on GGH15 key exchange does not explicitly compute encodings of zero as in the original exposition. Instead, the attack computes encodings of the same plaintext on two different source-to-sink paths (starting from different sources), and subtracts the encodings. In our setting we enforce without loss of generality that all graphs must have a single source, which can be generically achieved by connecting a “super” source node to the original source nodes of the graph, and encoding a 1 (or identity matrix) on edges leading into the original sources.

A diagram of our resulting graph is available in Appendix B.1, Figure 1. Note that we restrict attention to the 3-party key exchange construction and attack, as the ideas easily generalize to more parties.

The encodings used in the key exchange are $\mathbf{C}_{i,0}$ for $1 \leq i \leq 3$ (which we introduce to connect the super source node) and $\mathbf{C}_{i,i',l}$ for $1 \leq i, i' \leq 3, 1 \leq l \leq N$ (for some large enough N). Then for $\{\mathbf{C}\} = \{\mathbf{C}_{i,0}\}_{i \in \{1,2,3\}} \cup \{\mathbf{C}_{i,i',l}\}_{i,i' \in \{1,2,3\}, l \in [N]}$, the polynomial

$$p_{j,k}(\{\mathbf{C}\}) = \mathbf{C}_{2,0} \cdot \mathbf{C}_{2,1,1} \cdot \mathbf{C}_{2,2,j} \cdot \mathbf{C}_{2,3,k} - \mathbf{C}_{3,0} \cdot \mathbf{C}_{3,1,k} \cdot \mathbf{C}_{3,2,1} \cdot \mathbf{C}_{3,3,j}$$

is an encoding of $s_{3,1} \cdot s_{1,j} \cdot s_{2,k} - s_{2,k} \cdot s_{3,1} \cdot s_{1,j} = 0$ for all choices of $j \in [J], k \in [K]$, where for this attack $J = K = N$ (N is a parameter in the key exchange construction). Recall the key exchange construction uses a GGH15 variant that supports a commutative plaintext space, so this is always an encoding of 0. These details can be verified in Appendix B.1, but we stress they are not important for understanding how the attack fits in our model. Recall the key exchange construction uses a GGH15 variant that supports a commutative plaintext space, so this is always an encoding of 0.

- For the CGH17 attack, the source-to-sink encodings of zero are the results of branching program evaluations. Specifically, the branching program must have an input partition. Roughly speaking, the h branching program layers can be partitioned as $[h] = \mathcal{X} \parallel \mathcal{Z}$ where the matrices chosen in layers \mathcal{X} can be independently varied from the matrices chosen in layers \mathcal{Z} so that the result of the program is always 0. The encodings in this scheme are of the form $\mathbf{C}_{i,b}$ (corresponding to the “functional” branch of the GGHRSTW obfuscator) and $\mathbf{C}'_{i,b}$ (for the “dummy” branch), for $i \in [h], b \in \{0,1\}$. We also introduce bookend encodings $\mathbf{C}_0, \mathbf{C}'_0, \mathbf{C}_{h+1}, \mathbf{C}'_{h+1}$.

Following the notation of [19], we evaluate the branching program on all inputs $u^{(j,k)} \in \{0,1\}^\ell$ (ℓ denoting the length of the branching program inputs) where the input-partitioning guarantee is that the branching program evaluates to 0 on any choice of j, k for $j \in [J], k \in [K]$.

This gives us our zero-test polynomials $p_{j,k}$. Let $\text{inp} : [h] \rightarrow [\ell]$ be the input selection function of the branching program, and let $u_{\text{inp}(i)}^{(j,k)}$ be the bit of $u^{(j,k)}$ read on the i -th layer of branching program evaluation. Then for $\{\mathbf{C}\} = \{\mathbf{C}_0, \mathbf{C}'_0, \mathbf{C}_{h+1}, \mathbf{C}'_{h+1}\} \cup \{\mathbf{C}_{i,b}, \mathbf{C}'_{i,b}\}_{i \in [h], b \in \{0,1\}}$, the input-partitioning of the branching program guarantees

$$\begin{aligned} p_{j,k}(\{\mathbf{C}\}) = & \mathbf{C}_0 \cdot \prod_{i \in \mathcal{X}} \mathbf{C}_{i,u_{\text{inp}(i)}^{(j,k)}} \cdot \prod_{i \in \mathcal{Z}} \mathbf{C}_{i,u_{\text{inp}(i)}^{(j,k)}} \cdot \mathbf{C}_{h+1} \\ & - \mathbf{C}'_0 \cdot \prod_{i \in \mathcal{X}} \mathbf{C}'_{i,u_{\text{inp}(i)}^{(j,k)}} \cdot \prod_{i \in \mathcal{Z}} \mathbf{C}'_{i,u_{\text{inp}(i)}^{(j,k)}} \cdot \mathbf{C}'_{h+1}, \end{aligned}$$

corresponds to a source-to-sink encoding of zero for all $j \in [J], k \in [K]$. Here, J and K parameters derive from the size of the input partition. Again, we emphasize that a thorough understanding of input partitioning or the above expression is not crucial to understanding our model. An interested reader can find a description of the input partitioning requirement and further details in the attack of Chen, Gentry, and Halevi [19]. A diagram is available in Appendix B.2, Figure 2.

- The CVW18 attack essentially relies on the same input-partitioning requirements of the CGH17 attack, with the difference being that the attacked obfuscator is no longer evaluated by computing two separate programs and subtracting them. For $\{\mathbf{C}\} = \{\mathbf{C}_{i,b}\}_{i \in [h], b \in \{0,1\}}$, the evaluations are roughly of the form

$$p_{j,k}(\{\mathbf{C}\}) = \prod_{i \in \mathcal{X}} \mathbf{C}_{i,u_{\text{inp}(i)}^{(j,k)}} \cdot \prod_{i \in \mathcal{Z}} \mathbf{C}_{i,u_{\text{inp}(i)}^{(j,k)}}.$$

A diagram and brief review of the CVW18 attack is available in Appendix B.3, Figure 3.

Step 2: Zero-Test and Build \mathbf{W} Matrix. Zero-test each of these top-level encodings, and let the result of zero-testing $p_{j,k}(\{\mathbf{C}\})$ be $T_{j,k}$. Construct a $J \times K$ matrix \mathbf{W} where the (j, k) -th entry $W_{j,k}$ is derived from $T_{j,k}$. In all current attacks, the matrix \mathbf{W} has the following properties:

- \mathbf{W} factors into $\mathbf{X} \times \mathbf{Y}$ where the rows of \mathbf{Y} are linearly independent over the integers (with high probability).
- There exists a column of \mathbf{X} that is in the column space of a $J \times \eta$ dimensional matrix \mathbf{M} , for some η that we specify below for each attack. Each entry of \mathbf{M} is a polynomial over the entries of pre-encoding matrices $\{\mathbf{S}\}$.

We describe the zero-test procedure and specific structure of these matrices in each of the attacks above:

- In the CLLT16 setting (augmented with our “super” source \mathcal{S}), we zero-test by multiplying $\mathbf{A}_{\mathcal{S}}$ with $p_{j,k}(\{\mathbf{C}\})$ evaluated over the encodings. This gives a zero-test result $T_{j,k}$ as a vector. Coron et al. observe that the first element of this vector can be written as a dot product $\mathbf{x}_j \cdot \mathbf{y}_k$ where the entries of \mathbf{x}_j depend only on the encodings corresponding to user 1 (and the fixed encodings) and the entries of \mathbf{y}_k depend only on the encodings corresponding to user 2 (and the fixed encodings). Moreover, the first element of \mathbf{x}_j is the pre-encoding $s_{1,j}$. Coron et al. also argue that arranging many column vectors \mathbf{y}_k into a square matrix \mathbf{Y} results in \mathbf{Y} being invertible with high probability. Thus we take $W_{j,k}$ to be the first element of $T_{j,k}$, \mathbf{X} to consist of the row vectors $\mathbf{x}_1, \dots, \mathbf{x}_J$, and \mathbf{M} to simply be the column vector $[s_{1,1} \ s_{1,2} \ \dots \ s_{1,J}]^{\top}$ (of dimension $J \times \eta$ where $\eta = 1$).
- In the CGH17 setting, since the attack targets the GGH15 with safeguards scheme [23], there are “GGH15 bookends” (as opposed to branching program bookends) \mathbf{j} and \mathbf{l} . Zero-testing works by computing the scalar $T_{j,k} = \mathbf{j} \cdot p_{j,k}(\{\mathbf{C}_{i,b}\}) \cdot \mathbf{l}$. We let $\mathbf{W}_{j,k} = T_{j,k}$. Chen et al. observe that $T_{j,k}$ can be written as the dot product $\mathbf{x}_j \cdot \mathbf{y}_k$ where \mathbf{x}_j only depends on encodings associated with the \mathcal{X} part of the input partition and \mathbf{y}_k only depends on encodings associated with the \mathcal{Z} part of the input partition. Say that each pre-encoding matrix $\mathbf{S}_{i,b}$ has dimension $w \times w$. Then the first w elements of \mathbf{x}_j is given by the vector

$$\mathbf{j} \cdot \prod_{i \in \mathcal{X}} \mathbf{S}_{i,u_{\text{inp}(i)}^{(j,:)}}.$$

And the very first element of \mathbf{x}_j is the scalar dot product

$$\mathbf{j} \cdot (\prod_{i \in \mathcal{X}} \mathbf{S}_{i,u_{\text{inp}(i)}^{(j,:)}})^{(1)}$$

where $\mathbf{S}^{(1)}$ denotes the first column of a matrix \mathbf{S} . Then if we again take \mathbf{X} to consist of the row vectors $\mathbf{x}_1, \dots, \mathbf{x}_J$,

$$\mathbf{X}^{(1)} = \begin{bmatrix} (\prod_{i \in \mathcal{X}} \mathbf{S}_{i,u_{\text{inp}(i)}^{(1,:)}})^{(1)\top} \\ \vdots \\ (\prod_{i \in \mathcal{X}} \mathbf{S}_{i,u_{\text{inp}(i)}^{(J,:)}})^{(1)\top} \end{bmatrix} \cdot \mathbf{j}^{\top}$$

So we let \mathbf{M} be the matrix specified above (which is of dimension $J \times \eta$ for $\eta = w$), which consists of all pre-encoding elements, and conclude that \mathbf{X} does indeed contain a column in the column space of \mathbf{M} . Finally, we remark that analysis from CGH17 shows that the matrix \mathbf{Y} consisting of the column vectors \mathbf{y}_k will be full rank with high probability.

- Zero-testing in the CVW18 setting with single-input 2-partition branching programs is identical to the CGH17 setting and the resulting matrix \mathbf{W} decomposes in the same way. The other settings of CVW18 follow analogously.

Step 3: Deriving an Algebraic Relation. At this point, the CLLT16, CGH17, and CVW18 attacks use different strategies to mount an attack starting from the \mathbf{W} matrix. To show all of these attacks are captured by our model, we will disregard the details of how the individual attacks use the \mathbf{W} matrix. Instead, we demonstrate that this \mathbf{W} matrix is already sufficient to come up with a Q satisfying our post-zero-test win condition (with non-negligible probability). The following steps are written generically, in that they apply to the \mathbf{W} matrix generated in each of the CLLT16, CGH17, and CVW18 attacks. It suffices to give a polynomial-time procedure (which we refer to as the adversary) that extracts a Q satisfying our win condition.

To win in our model, the adversary will pick the parameter K so that \mathbf{Y} turns out to be square and thus invertible and the parameter $J \geq K + \eta$ (where η is specified in step 2 by the setting we are in). \mathbf{Y} being invertible implies that every column of \mathbf{X} is in the column space of \mathbf{W} , so in particular we have a column of \mathbf{X} that is in both the column space of \mathbf{W} and the column space of \mathbf{M} . Intuitively, if we are able to combine the columns of \mathbf{W} and \mathbf{M} into a square matrix, we are guaranteed that the determinant of this matrix will be zero. We just have to ensure that the columns from \mathbf{W} and the columns from \mathbf{M} are each linearly independent so that the determinant polynomial is not identically zero when either set of variables is substituted in. The adversary mounts the attack as follows, where the parameter β is taken to be exponential in the security parameter λ that the underlying scheme was initialized with, and \leftarrow denotes “drawing uniformly at random.”

To start, the adversary forms the matrix \mathbf{W} of handles to honest zero-test results and the matrix \mathbf{M} of pre-encoding handles where $\mathbf{W} \in \mathbb{Z}^{J \times K}$ and $\mathbf{M} \in \mathbb{Z}^{J \times \eta}$. The adversary then guesses the ranks $r_{\mathbf{M}}$ of \mathbf{M} and $r_{\mathbf{W}}$ of \mathbf{W} uniformly at random. The adversary guesses the correct ranks with probability $1/(K\eta)$.

The adversary then draws four random matrices $\mathbf{U}, \mathbf{U}' \leftarrow \mathbb{Z}_{\beta}^{(r_{\mathbf{M}}+r_{\mathbf{W}}) \times J}, \mathbf{V} \leftarrow \mathbb{Z}_{\beta}^{\eta \times r_{\mathbf{M}}}, \mathbf{V}' \leftarrow \mathbb{Z}_{\beta}^{K \times r_{\mathbf{W}}}$, and constructs

$$\mathbf{M}' = \mathbf{U} \cdot \mathbf{M} \cdot \mathbf{V}, \text{ and } \mathbf{W}' = \mathbf{U}' \cdot \mathbf{W} \cdot \mathbf{V}'.$$

Note that $\mathbf{M}' \in \mathbb{Z}^{(r_{\mathbf{M}}+r_{\mathbf{W}}) \times r_{\mathbf{M}}}$, and $\mathbf{W}' \in \mathbb{Z}^{(r_{\mathbf{M}}+r_{\mathbf{W}}) \times r_{\mathbf{W}}}$. Lastly, the adversary constructs a square $(r_{\mathbf{M}} + r_{\mathbf{W}}) \times (r_{\mathbf{M}} + r_{\mathbf{W}})$ matrix $\mathbf{A} = [\mathbf{M}' \mid \mathbf{W}']$ by concatenating \mathbf{M}' and \mathbf{W}' . Note that the entries of \mathbf{A} are over handles to the zero-test results and the pre-encodings. The adversary takes the determinant polynomial Q of this matrix and submits Q as the post-zero-test polynomial.

Assume the adversary has guessed the two ranks correctly, which happens with non-negligible probability since $K, \eta = \text{poly}(\lambda)$. We now show that Q will satisfy the following three win conditions in our model with non-negligible probability.

1. $Q(\{T_{j,k}\}, \{S_{i,j,k}\}) = 0$
2. $Q(\{T_{j,k}\}, \{\widehat{S}_{i,j,k}\}) \not\equiv 0$
3. $Q(\{\widehat{T}_{j,k}\}, \{S_{i,j,k}\}) \not\equiv 0$

First, $Q(\{T_{j,k}\}, \{S_{i,j,k}\}) = 0$ since we have explicitly introduced a linear dependency among the columns of \mathbf{A} . Now we argue that with high probability, \mathbf{M}' has an $r_{\mathbf{M}} \times r_{\mathbf{M}}$ dimensional submatrix of rank $r_{\mathbf{M}}$ which implies that its columns are linearly independent and thus that $Q(\{\widehat{T}_{j,k}\}, \{S_{i,j,k}\}) \not\equiv 0$. The same argument applies to \mathbf{W}' implying that $Q(\{T_{j,k}\}, \{\widehat{S}_{i,j,k}\}) \not\equiv 0$. This follows from an application of the following lemma, noting that in our case, β is exponential in λ and the dimensions of \mathbf{M} and \mathbf{W} are polynomial in λ .

Lemma 4. Suppose an $\mathbf{M} \in \mathbb{Z}_{\beta}^{n \times m}$ has rank r . Draw uniformly random $\mathbf{U} \leftarrow \mathbb{Z}_{\beta}^{r \times n}, \mathbf{V} \leftarrow \mathbb{Z}_{\beta}^{m \times r}$. Then $\mathbf{M}' := \mathbf{U} \cdot \mathbf{M} \cdot \mathbf{V}$ is full rank with probability at least $1 - \frac{2r}{\beta}$.

Proof. Let the entries of \mathbf{U} and \mathbf{V} be formal variables and view \mathbf{M}' as a matrix over these variables. We know that there exists an $r \times r$ full rank submatrix of \mathbf{M} . Now consider the determinant of the matrix \mathbf{M}' , which is a degree- $2r$ polynomial over the \mathbf{U} and \mathbf{V} variables. We can imagine setting the variables to 0/1 so that

this submatrix is exactly equal to the full rank submatrix in \mathbf{M} , implying that the determinant polynomial is not identically zero. Now by the Schwartz-Zippel lemma, this polynomial will be non-zero (implying that \mathbf{M}' is full rank) with probability $\geq 1 - \frac{2r}{\beta}$ over the random choice \mathbf{U} and \mathbf{V} since they came from a subset of the rational numbers of size β . \square

3.6 Limitations of Our Model

Our model does not permit a number of common operations that might arise in standard lattice cryptanalysis. For example, we naturally disallow any modular reductions or rounding on the results of zero-testing, since the relation would no longer be algebraic. This may at first appear problematic, since it means our model does not capture many simple attack strategies such as LLL [33].

We stress, however, that this is a common feature of many abstract attack models defined in the literature. For example, the random oracle model does not allow for differential cryptanalysis, despite it being a powerful way to attack hash functions. This is usually considered okay, since schemes are tuned (say, by increasing the number of rounds) to make such attacks useless. Similarly, the generic group model is often applied to elliptic curves, even though the model does not allow for known attacks such as the MOV attack [34]. Instead, these models capture things the adversary can do no matter how parameters are chosen.

Our setting is similar, as most lattice attacks can be defeated by tuning parameters. The most devastating attacks on schemes such as GGH15 are zeroizing attacks, as they are present *no matter how parameters are chosen*. Therefore, we devise a model that accurately captures how zeroizing attacks are performed, and tune parameters to block all other attacks.

4 Towards Zeroizing Resistance: New Models and Constructions

4.1 Section Overview

In this section we construct a graph-induced encoding scheme with two desirable properties.

Property 1: Asymmetric Levels. In asymmetric multilinear maps such as GGH13 and CLT13, plaintexts are encoded relative to subsets $\ell \subseteq [\kappa]$, where κ is a positive integer. Two encodings can be added if and only if they are encoded at the same level set and can be multiplied if and only if they are encoded at disjoint level sets. Only top level $[\kappa]$ encodings can be zero-tested. In certain settings such as obfuscation, it is desirable to enforce restrictions based on these asymmetric levels (for example, to implement straddling sets which prevent “mixed-input” attacks [22, 24]). Unfortunately, the GGH15 edge restrictions do not immediately give us the same capabilities of asymmetric level restrictions. Thus, we require a notion of “Graph-Induced Multilinear Maps with Asymmetric Levels”, which simultaneously associates every encoding with a graph path $u_i \rightsquigarrow v_i$ as well as a level set $\ell \subseteq [\kappa]$ (first described by Halevi [23]). Addition, multiplication, and zero-test operations are only allowed as long as both the graph-induced restrictions and the asymmetric level set restrictions are satisfied.

We naturally redefine our GGH15 Zeroizing Model for this new notion, calling the resulting model the “Level-Restricted GGH15 Zeroizing Model”. This model is identical to the GGH15 Zeroizing Model, except the adversary is now forced to additionally respect the asymmetric level restrictions when computing a top-level encoding of zero.

Property 2: Semantic Security of Encodings. Recent techniques of Chen et al. [9] show how to produce GGH15 encodings that achieve provable semantic security from LWE via a new construction they call “ γ -GGH15 encodings”. Note that this semantic security guarantee is orthogonal to what our GGH15 Zeroizing Model captures. Semantically secure encodings ensure that the encodings themselves do not leak information, but only in the setting where successful zero-tests are computationally unachievable. On the other hand, our GGH15 Zeroizing Model captures adversaries who attack using the zero-test leakage but only under the idealized assumption that the encodings themselves leak nothing.

A New GGH15 Variant We integrate these two new techniques into a new construction we call γ -GGH15-AL (γ -encodings and asymmetric levels). We enforce asymmetric levels using a simple trick of dividing by random scalars due to Halevi [23]. We show that security of our γ -GGH15-AL construction in the GGH15 Zeroizing Model implies security in a (more restrictive) Level-Restricted GGH15 Zeroizing Model. In other words, we prove that an attack on γ -GGH15-AL that is free to disobey the asymmetric level restrictions has no more power than an attack that obeys the asymmetric level restrictions. The proof proceeds from applications of the Schwartz-Zippel lemma, which allow us to argue that a top-level encoding that disobeys level restrictions will not give a successful zero-test (with overwhelming probability). To achieve semantic security guarantees, we incorporate the γ -GGH15 encoding strategy of [9] into our γ -GGH15-AL construction.

We note that semantic security is only a heuristic statement in our setting. The semantic security proofs of [9] hold when the adversary cannot successfully zero-test, but in our construction, zero-testing can be achieved using a right bookend vector. Thus, our construction only has semantic security when this bookend vector is hidden from the adversary. The intuition is that when the right bookend vector is not hidden, security is lost because of zeroizing attacks, at which point we appeal to our GGH15 Zeroizing Model.

At the end of this section, we introduce a third model we call the “GGH15 Annihilation Model.” We show that any successful zeroizing attacks in the GGH15 Zeroizing Model on our γ -GGH15-AL construction imply the existence of a successful adversary in the GGH15 Annihilation Model (by first going through the Level-Restricted GGH15 Zeroizing Model). An adversary in the GGH15 Annihilation Model will correspond to a polynomial-complexity arithmetic circuit that *annihilates* the zero-test polynomials submitted by the adversary.

Looking ahead to Section 5, these proofs will allow us to construct a new obfuscation candidate based on our γ -GGH15-AL map and easily prove it secure in the GGH15 Zeroizing Model. We will show that any adversary that mounts a successful zeroizing attack on our candidate in the GGH15 Zeroizing Model will imply a successful adversary in the GGH15 Annihilation Model. For our particular obfuscation candidate, a successful adversary in the GGH15 Annihilation Model will violate the Branching Program Un-Annihilability Assumption of Garg et al. [20]. Thus, we will obtain an obfuscation candidate with provable resistance against zeroizing attacks, coupled with semantically secure encodings (when the right bookend is hidden).

4.2 γ -GGH15 Encodings

Our construction will use the γ -GGH15 encodings (also known as generalized GGH15 encodings) introduced by Chen et al. [9]. The purpose of these encodings is to achieve semantic security from LWE when the adversary cannot obtain successful zero-tests. Chen et al. [9] only consider the setting of matrix branching programs, and their definitions are with respect to a path graph with exactly two matrices encoded on each edge. We note that it is not too difficult to extend their definitions and security guarantees to arbitrary one-source/one-sink directed acyclic graphs (DAGs) with any polynomial number of matrices encoded on each edge. For a full description of the γ -GGH15 encodings as defined for encodings of matrix branching programs, refer to [9].

Consider the functions⁷

$$\gamma_{\text{diag}}(\mathbf{M}, \mathbf{S}) = \begin{bmatrix} \mathbf{M} \\ \mathbf{S} \end{bmatrix}, \quad \gamma_{\otimes \text{diag}}(\mathbf{M}, \mathbf{S}) = \begin{bmatrix} \mathbf{M} \otimes \mathbf{S} \\ \mathbf{S} \end{bmatrix}.$$

Notation. Let the nodes of the directed acyclic graph be labeled in topological order as $i = 1, \dots, d$. To support a general one-source/one-sink DAG, we allow encoding on any directed edge (i, i') where $i < i'$ and $i, i' \in \{1, \dots, d\}$. Let the number of plaintexts encoded on edge (i, i') be denoted $N(i, i')$. We index the encodings on edge (i, i') with the variable j , so that $\{\mathbf{D}_{(i, i'), j}\}_{j \in [N(i, i')]}^{} \cup \{\mathbf{D}_{(i, i'), 0}\}^{} \cup \{\mathbf{D}_{(i, i'), -1}\}^{} \cup \{\mathbf{D}_{(i, i'), -2}\}^{} \cup \dots$ is the set of all $N(i, i')$ encodings on edge (i, i') . Finally, let $E = \max_i \{\sum_{i'} N(i, i')\}$ be the maximum number of encodings relative to any source vertex, and let $M = \sum_{i, i'} N(i, i')$ be the total number of encodings we consider. To recover the matrix branching program setting considered by Chen et al [9], simply set $N(i, i') = 2$ whenever $i' = i + 1$, and $N(i, i') = 0$ otherwise.

⁷ Other γ functions are considered in [9], but γ_{diag} and $\gamma_{\otimes \text{diag}}$ suffice for our purposes.

In this notation, we give a more general statement of Theorem 5.7 of [9], and give the proof in Appendix C. Here we only prove semantic security for γ_{diag} encodings, as the semantic security for $\gamma_{\otimes \text{diag}}$ naturally follows per Corollary 5.9 of [9]. We state the full theorem here re-written in our notation for completeness but note that the precise details are not essential for our construction and analysis.

Theorem 1 (Semantic Security of γ_{diag} Encodings, adapted from Theorem 5.7 of [9]). *Assuming $\text{LWE}_{n, Em, q, U(\mathbb{Z}_q), D_{\mathbb{Z}, \sigma}, D_{\mathbb{Z}, \sigma}}$, the following two distributions are computationally indistinguishable:*

$$\begin{aligned} & \mathbf{J} \cdot \mathbf{A}_1, \{\mathbf{D}_{(i, i'), j}, \mathbf{P}_{(i, i'), j}, \mathbf{M}_{(i, i'), j}\}_{i < i', j \in [N(i, i')]} , \overline{\mathbf{A}}_{d+1} \\ & \approx_c \\ & \mathbf{J} \cdot \mathbf{A}_1, \{\mathbf{V}_{(i, i'), j}, \mathbf{P}_{(i, i'), j}, \mathbf{M}_{(i, i'), j}\}_{i < i', j \in [N(i, i')]} , \overline{\mathbf{A}}_{d+1} \end{aligned}$$

where

- $\{\mathbf{A}_i, \tau_i \leftarrow \text{TrapSam}(1^n, 1^m, q)\}_{i \in \{1, \dots, d+1\}}, \mathbf{A}_d \leftarrow U(\mathbb{Z}_q^{n \times m}), \mathbf{J} \in \{0, 1\}^{n' \times (n-n')} \mid \mathbf{I}^{n' \times n'}$.
- For any matrix $\mathbf{X} \in \mathbb{Z}^{n \times *}$, $\overline{\mathbf{X}}$ denotes the first $(n - n')$ rows of \mathbf{X} , and $\underline{\mathbf{X}}$ denotes the last n' rows of \mathbf{X} .
- $\mathbf{P}_{(i, i'), j} \leftarrow D_{\mathbb{Z}, \sigma}^{n' \times n'}, \{\mathbf{M}_{(i, i'), j}\}_{i < i', j \in [N(i, i')]} \leftarrow f(\{\mathbf{P}_{i, i'}, j\}_{i < i', j \in [N(i, i')]})$ for $f : (\mathbb{Z}^{n' \times n'})^M \rightarrow (\mathbb{Z}^{(n-n') \times (n-n')})^M$.
- $\mathbf{D}_{(i, i'), j} \leftarrow \mathbf{A}_i^{-1} \begin{bmatrix} \mathbf{M}_{(i, i'), j} \mathbf{A}_{i'} + \mathbf{E}_{(i, i'), j} \\ \mathbf{P}_{(i, i'), j} \mathbf{A}_{i'} + \underline{\mathbf{E}}_{(i, i'), j} \end{bmatrix}, \mathbf{E}_{(i, i'), j} \leftarrow \chi^{n \times m}$.
- $\mathbf{V}_{(i, i'), j} \leftarrow D_{\mathbb{Z}, \sigma}^{m \times m}$.

4.3 A Graph-Induced Encoding Scheme with Asymmetric Levels

Overview To encode a plaintext matrix \mathbf{M} on an edge $i \rightsquigarrow j$ with level set $L \subseteq [\kappa]$ we first generate a random matrix \mathbf{P} in order to apply the $\gamma_{\otimes \text{diag}}$ function of [9]. The resulting pre-encoding $\text{diag}(\mathbf{M} \otimes \mathbf{P}, \mathbf{P})$ is encoded via the ordinary GGH15 encoding procedure to obtain an encoding \mathbf{D} . The next step is to draw a random $k \times k$ matrix \mathbf{B} and append it on along the diagonal. This matrix \mathbf{B} ensures each final encoding matrix \mathbf{C} has sufficient entropy (used in Lemma 6), and is crucial for Lemma 7. The next step is to multiply by Kilian-randomization matrices (drawn by KeyGen for each vertex), and then divide by level scalars $\prod_{\ell \in L} z_\ell$. The resulting encoding is

$$\mathbf{C} = \left(\prod_{\ell \in L} z_\ell \right)^{-1} \cdot \mathbf{R}_i^{-1} \cdot \begin{bmatrix} \mathbf{D} & \\ & \mathbf{B} \end{bmatrix} \cdot \mathbf{R}_j.$$

To ensure that zero-testing works, we construct our right bookend vector \mathbf{w} to contain the product $(\prod_{\ell \in [\kappa]} z_\ell)$, which cancels out the level scalars in the encoding as long as it is at the top level $[\kappa]$. The left and right bookends also contain Kilian-randomization matrices \mathbf{R}_1 and \mathbf{R}_d^{-1} multiplied in to cancel out the Kilian-randomization on the encodings. The bookends contain additional components \mathbf{b}_v and \mathbf{b}_w^\top which multiply with the \mathbf{B} random matrices during zero-testing. This has the effect of adding the products of random matrices (with two random bookends) to the result of any zero-test (this will be crucial for our obfuscation security proof, where it will have the effect of adding a random branching program evaluation). The remaining bookend components are essentially set to be the bookends required by the γ -GGH15 encodings. However, we also multiply them by randomly sampled vectors \mathbf{v}' and \mathbf{w}' to simplify dimensions.

Construction $\gamma\text{-GGH15-AL.KeyGen}(1^\lambda, G, R = \mathbb{Z}, \kappa, \beta, k)$:⁸

Parameter Generation

⁸ κ is the number of asymmetric levels, β is a bound on the size of plaintext entries, and k is the dimension of the block diagonal matrices we append during the encoding procedure.

- Label the nodes of G in topological order as $1, \dots, d$ where node 1 is the unique source and node d is the unique sink.
- Choose parameters $n, w, n', m, q, \sigma, \chi, B$ where $n = wn' + n'$ according to the remark below. All operations happen over \mathbb{Z}_q . Plaintexts have dimension $w \times w$ with entries bounded by β , pre-encodings have dimension $n \times n$ with entries bounded (with high probability) by $\beta \cdot \sigma \cdot \sqrt{n}$, and encodings have dimension $(m+k) \times (m+k)$ with entries bounded by $\nu = 2^\lambda$. We draw error matrices under distribution $(\chi)^{n \times m}$ and set B to be the zero-test bound.

Instance Generation

- (GGH15 matrices and trapdoors) For each vertex $i \in V$, sample $(\mathbf{A}_i, \tau_i) \leftarrow \text{TrapSam}(1^n, 1^m, q)$.
- (Kilian-randomization matrices) For each vertex $i \in V$, sample a random invertible $\mathbf{R}_i \in \mathbb{Z}_q^{(m+k) \times (m+k)}$.
- (Asymmetric level scalars) For each level $\ell \in [\kappa]$, sample a random invertible $z_\ell \in \mathbb{Z}_q$.

Bookend Generation

- (Left bookend matrix from γ -GGH15 encodings) Sample a random $\mathbf{J}' \leftarrow \{0, 1\}^{n' \times wn'}$ and define

$$\mathbf{J} := [\mathbf{J}' \mid \mathbf{I}^{n' \times n'}].$$

- (Encoding matrix used in right bookend) Sample a uniform $\mathbf{A}^* \leftarrow \mathbb{Z}_q^{n \times m}$, an error matrix $\mathbf{E}^* \leftarrow (\chi)^{n \times m}$, and compute

$$\mathbf{D}^* \leftarrow \text{SampleD}(\mathbf{A}_d, \tau_d, \begin{bmatrix} \mathbf{I}^{wn' \times wn'} \\ \mathbf{0}^{n' \times n'} \end{bmatrix} \cdot \mathbf{A}^* + \mathbf{E}^*, \sigma)$$

This encoding serves to cancel out the lower random block diagonals on pre-encodings and enables zero-testing on the actual plaintexts.

- (Random bookend vectors) Sample $\mathbf{v}' \leftarrow D_{\mathbb{Z}, \sigma}^{n'}, \mathbf{w}' \leftarrow D_{\mathbb{Z}, \sigma}^m$.
- (Final bookend vectors) Sample uniform $\mathbf{b}_v \in \mathbb{Z}_\nu^k, \mathbf{b}_w \in \mathbb{Z}_\nu^k$ and compute the final bookends

$$\mathbf{v} = [\mathbf{v}' \cdot \mathbf{J} \cdot \mathbf{A}_1 | \mathbf{b}_v] \cdot \mathbf{R}_1, \quad \mathbf{w} = (\prod_{\ell \in [\kappa]} z_\ell) \cdot \mathbf{R}_d^{-1} \cdot \begin{bmatrix} \mathbf{D}^* \cdot \mathbf{w}'^\top \\ \mathbf{b}_w^\top \end{bmatrix}.$$

Output

- Public parameters $\text{pp} = \{n, w, n', m, k, q, \sigma, \chi, B, \mathbf{v}, \mathbf{w}\}$
- Secret parameters $\text{sp} = \{\mathbf{A}_i, \tau_i, \mathbf{R}_i\}_{i \in [d]}, \{z_\ell\}_{\ell \in [\kappa]}$

γ -GGH15-AL.Enc($\text{sp}, \mathbf{M} \in \mathbb{Z}_\beta^{w \times w}, i \rightsquigarrow j, L \subseteq [\kappa]$):

- Draw $\mathbf{P} \leftarrow D_{\mathbb{Z}, \sigma}^{n' \times n'}$ and $\mathbf{E} \leftarrow (\chi)^{n \times m}$
- Compute $\mathbf{D} \leftarrow \text{SampleD}(\mathbf{A}_i, \tau_i, \begin{bmatrix} \mathbf{M} \otimes \mathbf{P} \\ \mathbf{P} \end{bmatrix} \cdot \mathbf{A}_j + \mathbf{E}, \sigma)$
- Draw uniform $\mathbf{B} \leftarrow \mathbb{Z}_\nu^{k \times k}$ and output the encoding

$$\mathbf{C} = (\prod_{\ell \in L} z_\ell)^{-1} \cdot \mathbf{R}_i^{-1} \cdot \begin{bmatrix} \mathbf{D} \\ \mathbf{B} \end{bmatrix} \cdot \mathbf{R}_j$$

γ -GGH15-AL.ZeroTest(pp, \mathbf{C}):

- Return zero if $|\mathbf{v} \cdot \mathbf{C} \cdot \mathbf{w}^\top| \leq B$, and not zero otherwise.

Parameters. First, we derive an additional security parameter $\lambda_{\text{LWE}} = \text{poly}(\lambda)$ which determines the hardness of LWE instances associated with the construction. We set the encoding bound $\nu = 2^\lambda$ and choose $n, w, n', m, q, \sigma, \chi = D_{\mathbb{Z}, s}$ where $n = wn' + n'$, $m = \Theta(n \log q)$ and $\sigma = \Theta(\sqrt{n \log q})$ for trapdoor functionality and $n' = \Theta(\lambda_{\text{LWE}} \log q)$ and $s = \Omega(\sqrt{n'})$ for LWE security.⁹ Set the zero-test bound $B := (m \cdot \beta \cdot \sigma \cdot \sqrt{n})^{d+1} + (k \cdot \nu)^{d+1}$ and choose $q \geq B \cdot \omega(\text{poly}(\lambda))$ such that $q \leq (\sigma/\lambda_{\text{LWE}}) \cdot (2^{\lambda_{\text{LWE}}})^{1-\epsilon}$ for some $\epsilon \in (0, 1)$.

We briefly argue that these constraints can be satisfied with $\lambda_{\text{LWE}} = \text{poly}(\lambda)$. First, we can take $\log q = \Theta(\lambda_{\text{LWE}})$ and still satisfy the final constraint. Then since $w, k, d = \text{poly}(\lambda)$, we can write B as $\lambda_{\text{LWE}}^{\text{poly}(\lambda)}$. Now the remaining constraint is $2^{\Theta(\lambda_{\text{LWE}})} \geq \lambda_{\text{LWE}}^{\text{poly}(\lambda)} \cdot \omega(\text{poly}(\lambda))$, which is satisfied by taking λ_{LWE} to be a sufficiently large polynomial in λ .

We now argue that this setting of parameters enables zero-test functionality. Say that we test a source-to-sink encoding \mathbf{C} . If \mathbf{C} is an encoding of $\mathbf{0}$, then in the worst case (for noise growth) it will be the product of $d - 1$ encodings \mathbf{D}_i of plaintexts \mathbf{M}_i with error matrices \mathbf{E}_i . In this case (seen by simply expanding out GGH15 multiplications),

$$\begin{aligned} |\mathbf{v} \cdot \mathbf{C} \cdot \mathbf{w}^\top| &= |[\mathbf{v}' \cdot \mathbf{J} \cdot \mathbf{A}_1 | \mathbf{b}_v] \cdot \mathbf{C} \cdot [(\mathbf{D}^* \cdot \mathbf{w}'^\top)^\top | \mathbf{b}_w]^\top| \\ &= |\mathbf{v}' \cdot \mathbf{J} \cdot \sum_{j=1}^{d-1} ((\prod_{i=1}^{j-1} \begin{bmatrix} \mathbf{M}_i \otimes \mathbf{P}_i \\ \mathbf{P}_i \end{bmatrix}) \cdot \mathbf{E}_j \cdot \prod_{k=j+1}^{d-1} \mathbf{D}_k) \cdot \mathbf{D}^* \cdot \mathbf{w}'^\top + \mathbf{b}_v \cdot \prod_{i=1}^{d-1} \mathbf{B}_i \cdot \mathbf{b}_w^\top| \\ &\leq |\sigma^2 \cdot m^2 \cdot (m \cdot \beta \cdot \sigma \cdot \sqrt{n})^{d-1} + (k \cdot \nu)^{d+1}| \leq B \end{aligned}$$

And if \mathbf{C} is an encoding of some non-zero \mathbf{M} , this multiplication includes the term $\mathbf{M} \cdot \mathbf{A}^*$ for \mathbf{A}^* uniform in \mathbb{Z}_q and we have that $\frac{B}{q} = \text{negl}(\lambda)$.

Lemma 5 (Semantic Security). *Any set of encodings generated with γ -GGH15-AL achieves semantic security from LWE, assuming we release a modified set of public parameters $\text{pp}' = \text{pp} \setminus \{\mathbf{w}\}$ that hides the right bookend vector \mathbf{w} .*

Proof. This follows immediately from Theorem 1, noting that the modifications we make do not hurt security. Instead of giving out the exact $\mathbf{J} \cdot \mathbf{A}_1, \{\mathbf{D}_{(i,i'),j}\}_{i < i', j \in [N(i,i')]}$ terms in the statement of Theorem 1, our construction changes the following:

- A random vector is multiplied into $\mathbf{J} \cdot \mathbf{A}_1$.
- The \mathbf{D} matrices are divided by random scalars
- Random block diagonal \mathbf{B} matrices are appended to the \mathbf{D} matrices and the result is Kilian-randomized.

Each of the above steps is easily simulated and thus the semantic security of the distribution in Theorem 1 implies the semantic security of encodings produced by the above scheme. \square

4.4 Level-Restricted GGH15 Zeroizing Model

In order to define this model, we need the following definition.

Definition 4 (Level-Respecting Encodings). *Fix a universe of levels $[\kappa]$. Let L_i be the set of levels associated with encoding \mathbf{C}_i . Let m be a monomial over encodings $\{\mathbf{C}_i\}$ which contains the j encodings $\mathbf{C}_1, \dots, \mathbf{C}_j$. Then m is level-respecting if L_1, \dots, L_j are disjoint and $\bigcup_{i=1}^j L_i = [\kappa]$. A polynomial p over encodings $\{\mathbf{C}_i\}$ is level-respecting if and only if each of its monomials is.*

We only mention the differences between this model and the GGH15 Zeroizing Model. Here we expect that the GGH15 variant G that the model is initialized with supports asymmetric levels, namely that $\mathsf{G}.\text{Enc}$ additionally takes as input a level set $L \subseteq [\kappa]$.

⁹ Following Chen et. al. [9]

Initialize Parameters. The model \mathcal{M} in addition takes a parameter κ denoting the number of asymmetric levels.

Initialize Elements. \mathcal{M} is additionally given a level set $L_i \subseteq [\kappa]$ along with each plaintext \mathbf{M}_i and path $u_i \rightsquigarrow v_i$. \mathcal{M} computes the corresponding pre-encoding \mathbf{S}_i (from G.PreProcess), and computes the encoding

$$\mathbf{C}_i \leftarrow \text{G.Enc}(\mathbf{sp}, \mathbf{S}_i, u_i \rightsquigarrow v_i, L_i).$$

\mathcal{M} stores $(\mathbf{S}_i, \mathbf{C}_i, u_i \rightsquigarrow v_i, L_i)$ in a *pre-zero-test table*.

Zero-test. When the adversary submits a polynomial p , \mathcal{M} additionally checks that it is level-respecting, and if it is not, \mathcal{M} returns \perp .

Lemma 6. *Let \mathcal{A} be a successful adversary in the GGH15 Zeroizing Model instantiated with γ -GGH15-AL. Then there exists a successful adversary \mathcal{A}' in the Level-Restricted GGH15 Zeroizing Model instantiated with γ -GGH15-AL.*

Proof. We show that with overwhelming probability, every zero-test polynomial submitted by \mathcal{A} that does not result in \perp is already level-respecting, so any \mathcal{A} that wins in the GGH15 Zeroizing model also wins in the Level-Restricted GGH15 Zeroizing Model.

Consider any edge-respecting polynomial $p(\{\hat{\mathbf{C}}\})$ that \mathcal{A} submits for a successful zero-test. The result of zero-testing is the scalar element $\mathbf{v} \cdot p(\{\mathbf{C}\}) \cdot \mathbf{w}^\top$. The proof proceeds from evaluating this zero-test result $\mathbf{v} \cdot p(\{\mathbf{C}\}) \cdot \mathbf{w}^\top$ in two different ways.

Recall that the entries of the matrices $\{\mathbf{C}\}$ include random ‘‘level scalars’’ $\{z_j\}$. We imagine plugging in all of the actual values of every term in $\mathbf{v} \cdot p(\{\mathbf{C}\}) \cdot \mathbf{w}^\top$, except for the values of the random level scalars $\{z_j\}$, which we leave as formal variables $\{\hat{z}_j\}$. The result of this substitution is a rational function $h(\{\hat{z}_j\})$. By construction, h is constant over the \hat{z}_j variables if and only if p was level-respecting. Since p corresponds to a successful zero-test, we know that $|h(\{z_j\})| \leq B$ when the actual random scalars $\{z_j\}$ have been plugged in for $\{\hat{z}_j\}$. Equivalently, there exists some $\beta \in [-B, B]$ such that $h(\{z_i\}) - \beta = 0$. For a graph with d vertices and level set κ , $(\prod_{i \in [\kappa]} \hat{z}_i)^d (h(\{\hat{z}_j\}) - \beta)$ is a polynomial of degree $(\kappa + 1) \cdot d$ in the \hat{z}_j formal variables. We plug in the z_j variables and invoke the Schwartz-Zippel lemma to conclude that if this polynomial is not the identically-zero polynomial, it can only evaluate to zero with probability at most $(\kappa + 1)d/q$ over the randomness of the z_j values. Taking a union bound over all $2B$ possible values of β gives the final probability $\frac{2B}{q}(\kappa + 1)d = \text{negl}(\lambda)$.

This implies that the rational function $h(\{\hat{z}_j\})$ is a constant function and thus that each monomial containing any \hat{z}_j variables has coefficient 0. Now we can view any such coefficient as a polynomial in terms of the formal variables \hat{b}_k standing in for the elements of the random block diagonals \mathbf{B}_i added during encoding. Recall that these matrices were drawn uniformly from a space of size $\nu = 2^\lambda$ and the degree of the \hat{b}_k variables is bounded by $d = \text{poly}(\lambda)$. Applying the Schwartz-Zippel lemma again shows that this polynomial is identically zero over the \hat{b}_k variables with probability $1 - \text{negl}(\lambda)$. Since the block diagonals for each encoding are drawn independently, we conclude that with overwhelming probability, the coefficient of each monomial containing any \hat{z}_j variables is identically zero over the encoding matrices, implying that $p(\{\hat{\mathbf{C}}\})$ is level-respecting. \square

4.5 GGH15 Annihilation Model

We turn to describing a new model which has properties that are much easier to reason about when proving security. Instead of requiring the adversary to find an algebraic relation in the post-zero-test stage, we instead require the adversary to find an annihilating polynomial for the set of successful zero-test polynomials it previously obtained. More specifically, this polynomial must annihilate the zero-test polynomials when evaluated on square matrices of formal variables of some dimension k .

This k affects the difficulty of winning in the model, since matrices of larger dimension will be harder to annihilate. The advantage of having this model is that we have a notion of winning that corresponds more

directly to the underlying plaintexts encoded with the scheme. Namely, if we are able to encode plaintexts (taking advantage of asymmetric levels) in such a way that annihilating successful zero-test polynomials is hard, we can immediately obtain security in this model.

We describe the differences between this model and the Level-Restricted GGH15 Zeroizing Model. First, there is no computational bound on the adversary — it can submit as many zero-test queries as it wants and can take as much computation as it wants in the post-zero-test stage. However, each post-zero-test polynomial it submits must be implemented with a polynomial size circuit. The other modifications are described below.

Initialize Parameters. The model \mathcal{M} takes in an additional ‘tuning’ parameter k , which determines in some sense how strong the win condition will be.

Post-zero-test. At this point the adversary has submitted a set $\{p_u\}_u$ of successful zero-test polynomials which we associate with a set of formal variables $\{\widehat{p}_u\}_u$. The adversary now submits a *polynomial sized* circuit \bar{C} that implements a polynomial $\bar{Q}(\{\widehat{p}_u\}_u)$ over these formal variables. The model \mathcal{M} associates a set of $k \times k$ matrices $\{\widehat{\mathbf{C}}_i\}_i$ of formal variables with the set of encodings $\{\mathbf{C}_i\}_i$ and considers two additional k -dimensional vectors $\widehat{\mathbf{v}}$ and $\widehat{\mathbf{w}}$ of formal variables. Note that each individual entry of each of these matrices and vectors is a distinct formal variable. \mathcal{M} returns “Win” if the following hold:

1. The degree of \bar{Q} is $2^{o(\lambda)}$
2. $\bar{Q}(\{\widehat{p}_u\}_u) \not\equiv 0$
3. $\bar{Q}(\{\widehat{\mathbf{v}} \cdot p_u(\{\widehat{\mathbf{C}}_i\}_i) \cdot \widehat{\mathbf{w}}^\top\}_u) \equiv 0$

Lemma 7. Fix any $k \in \mathbb{N}$. Let \mathcal{A} be a successful adversary in the Level-Restricted GGH15 Zeroizing Model instantiated with γ -GGH15-AL where KeyGen receives the parameter k . Then there exists a successful adversary \mathcal{A}' in the GGH15 Annihilation Model with tuning parameter k .

Proof. Consider a successful adversary \mathcal{A} in the Level-Respecting GGH15 Zeroizing Model instantiated with γ -GGH15-AL where KeyGen receives the parameter k (determining the size of block diagonals added during encoding). We derive a successful adversary \mathcal{A}' in the GGH15 Annihilation Model with tuning parameter k . First we introduce some notation. Denote by $\{\mathbf{C}_i\}_i$ the set of encodings produced in this instantiation of the Level-Respecting GGH15 Zeroizing Model. Let $\{b_n\}_n$ be the set of individual elements of the vectors and matrices $\{\mathbf{b}_v, \{\mathbf{B}_i\}_i, \mathbf{b}_w^\top\}$ and let $\{\widehat{b}_n\}_n$ be the corresponding set of formal variables. Suppose \mathcal{A} submits the set of m polynomials $\{p_u(\{\widehat{\mathbf{C}}_i\}_i)\}_{u \in [m]}$ for zero-testing followed by a post zero-test polynomial Q . Let $p_u(\{\widehat{\mathbf{B}}_i\}_i)$ denote the u -th polynomial where each encoding handle $\widehat{\mathbf{C}}_i$ is replaced by its corresponding block diagonal handle $\widehat{\mathbf{B}}_i$. Let \widehat{T}_u denote the handle to the u -th zero-test result $T_u := \mathbf{v} \cdot p_u(\{\mathbf{C}_i\}_i) \cdot \mathbf{w}^\top$ and let $T'_u(\{b_n\}_n) := \mathbf{b}_v \cdot p_u(\{\mathbf{B}_i\}_i) \cdot \mathbf{b}_w^\top$ denote the result of evaluating p_u over the $\{\mathbf{B}_i\}_i$ matrices and multiplying the result on each side by the corresponding sections of the left and right bookends.

To complete the proof, we show the existence of a polynomial-size circuit \bar{C} that computes a degree $2^{o(\lambda)}$ polynomial \bar{Q} , derived from Q , such that

1. The degree of \bar{Q} is $2^{o(\lambda)}$
2. $\bar{Q}(\{\widehat{T}_j\}_{j \in [m]}) \not\equiv 0$
3. $\bar{Q}(\{T'_j(\{\widehat{b}_i\})\}_{j \in [m]}) \equiv 0$

Since each $\widehat{\mathbf{B}}_i$ is a $k \times k$ matrix of formal variables and \mathbf{b}_v and \mathbf{b}_w^\top are k -dimensional vectors of formal variables, the existence of such a \bar{Q} implies the existence of a successful adversary \mathcal{A}' in the GGH15 Annihilation Model with tuning parameter k . \mathcal{A}' simply submits the same m zero-test polynomials as \mathcal{A} and then iterates over all polynomial sized circuits until it finds one that causes the model to output “Win”.

Recall that Q is a polynomial over the zero-test handles $\{\widehat{T}_u\}_u$ and pre-encoding handles $\{\widehat{S}_{i,j,k}\}_{i,j,k}$ and that $Q(\{\widehat{T}_u\}_u, \{S_{i,j,k}\}_{i,j,k}) \not\equiv 0$. First we plug in the actual values of the pre-encodings to get a non-zero polynomial $Q'(\{\widehat{T}_u\}_u)$ over just the zero-test handles with degree at most $2^{o(\lambda)}$ (since the degree of Q must

have been $2^{o(\lambda)}$ if the adversary was successful). Next, we imagine plugging the actual values of each element of each encoding \mathbf{C}_i into the zero-test polynomials p_u while keeping the random block diagonal elements as formal variables. If we do the same for the zero-test vectors, then by construction, each $T_u = T'_u(\{b_n\}_n) + k_u$ for some constant k_u independent of the values of $\{b_n\}$. Now we can view Q' as a polynomial over the $\{\hat{b}_n\}_n$ formal variables by making the following substitution:

$$Q'(\{\hat{T}_u\}_u) = Q'(\{T'_u(\{\hat{b}_n\}_n) + k_u\}_u)$$

Now observe that

$$Q'(\{T'_u(\{b_n\}_n) + k_u\}_u) = Q(\{T_u\}_u, \{\mathbf{S}_i\}_i) = 0$$

by definition of our win condition. Furthermore, the degree of Q' over the $\{\hat{b}_n\}$ variables is $2^{o(\lambda)}$ since each p_u is a linear combination of source-to-sink monomials and thus has polynomial degree in each of the \hat{b}_n variables. Since each \hat{b}_n variable is drawn independently from a set of size $\nu = 2^\lambda$, we can apply the Schwartz-Zippel lemma to conclude that

$$Q'(\{T'_u(\{\hat{b}_n\}_n) + k_u\}_u) \equiv 0$$

with all but negligible probability. Now recall that $Q'(\{\hat{T}_u\}_u) \not\equiv 0$, which implies that $Q'(\{\hat{T}_u + k_u\}_u) \not\equiv 0$ for any constants k_u since the highest order term after expanding is exactly equal to $Q'(\{\hat{T}_u\}_u)$. Let $\bar{Q}(\{\hat{T}_u\}_u) = Q'(\{\hat{T}_u + k_u\}_u)$. We have just seen that \bar{Q} satisfies item (1) above. In addition,

$$\bar{Q}(\{T'_u(\{\hat{b}_n\}_u)\}_u) = Q'(\{T'_u(\{\hat{b}_n\}_n) + k_u\}_u) \equiv 0$$

So \bar{Q} satisfies the last two conditions. \bar{Q} is clearly computable by a polynomial sized circuit since it is the result of simply hard-coding values into certain inputs of Q and adding constants to other inputs. \square

5 An iO Candidate with Zeroizing Resistance

We show how to use the γ -GGH15-AL construction presented in Section 4.3 to construct a new candidate indistinguishability obfuscation (iO) scheme, which we can prove secure in the GGH15 Zeroizing Model.

We design our obfuscator to invoke the Branching Program Un-Annihilability (BPUA) Assumption of Garg et al. [20]. Roughly, this assumption states that no polynomial-size circuit can annihilate the evaluations of every matrix branching program, provided we consider branching programs whose input bits are read many times and in interleaved layers.

Thus, the first step of our obfuscator is to pad the input branching program in order to satisfy the requirement of the BPUA Assumption. To facilitate this, one of the inputs to our obfuscator is the parameter $t = t(\ell, \lambda) \geq 4\ell^4$ which specifies the minimum number of layers required. Note that the resulting padded program may have length greater than t , so we use a separate variable d to denote the actual length of the branching program after padding. We also enforce that each *pair* of input bits is read together in many layers, which is required to invoke the p -Bounded Speedup Hypothesis of [24].

To encode the matrices with γ -GGH15-AL, we pick asymmetric level sets from a straddling set system. The sets are assigned precisely to enforce that evaluations respect the input read structure of the padded branching program. The encoding edges are picked so that the branching program evaluations are naturally computed by traversing a path graph.

5.1 Construction

Input. The input to the obfuscator is the security parameter λ and a dual-input branching program BP (defined in Section 2.4) of length h , width w , and input length ℓ . BP consists of the matrices $\{\mathbf{M}_{i,b_1,b_2}\}_{i \in [h], b_1, b_2 \in \{0,1\}}$ and input selection function $\text{inp} : [h] \rightarrow [\ell] \times [\ell]$ which satisfies the following requirements:

- For each $i \in [h] : \text{inp}(i)_1 \neq \text{inp}(i)_2$, where $\text{inp}(i)_1, \text{inp}(i)_2$ denote the first and second slots of $\text{inp}(i)$, respectively.
- For each pair $j \neq k \in [\ell]$, there exists $i \in [h]$ such that $\text{inp}(i) \in \{(j, k), (k, j)\}$.

BP is evaluated on input $x \in \{0, 1\}^\ell$ by checking whether

$$\prod_{i \in [h]} \mathbf{M}_{i, x(i)} = 0^{w \times w}$$

where we abbreviate $x(i) := (x_{\text{inp}(i)_1}, x_{\text{inp}(i)_2})$.

Step 1: Pad the branching program. We pad the branching program with identity matrices until it has $d \geq t$ layers to ensure the following conditions:

- Each pair of input bits (j, k) is read in at least $4\ell^2$ different layers.
- There exist layers $i_1 < i_2 < \dots < i_t$ such that $\text{inp}(i_1)_1, \dots, \text{inp}(i_t)_1$ cycles t/ℓ times through $[\ell]$.

Step 2: Form straddling sets. For each input index $i \in [\ell]$, let r_i be the number of layers in which the bit i is read, and create a straddling set system with universe $\mathbb{U}^{(i)}$ and subsets $\{S_{j,b}^{(i)}\}_{j \in [r_i], b \in \{0,1\}}$. Let $\mathbb{U} := \bigcup_{i \in [\ell]} \mathbb{U}^{(i)}$.

Step 3: Encode with γ -GGH15-AL. Let G be a path graph with $d + 1$ nodes $1, \dots, d + 1$ and initialize the γ -GGH15-AL construction¹⁰

$$\text{pp}, \text{sp} \leftarrow \gamma\text{-GGH15-AL.KeyGen}(1^\lambda, G, \mathbb{Z}, |\mathbb{U}|, \max_{i, b_1, b_2} \{\|\mathbf{M}_{i, b_1, b_2}\|_\infty\}, k = 5).$$

For $i \in [d]$ and $b \in \{1, 2\}$, define $j_b(i)$ to be the number of times $\text{inp}(i)_b$ has been read after reading i columns of the branching program, and compute

$$\mathbf{C}_{i, b_1, b_2} \leftarrow \gamma\text{-GGH15.Enc}(\text{sp}, \mathbf{M}_{i, b_1, b_2}, i \rightsquigarrow i + 1, S_{j_1(i), b_1}^{\text{inp}(i)_1} \cup S_{j_2(i), b_2}^{\text{inp}(i)_2}).$$

5.2 Security

We immediately have that our obfuscation candidate satisfies semantic security without the right bookend by Lemma 5. However, the primary contribution of our obfuscator is to additionally give a formal proof of zeroizing resistance. Our main security theorem requires the p -Bounded Speedup Hypothesis of Miles et al. [24] and the Branching Program Un-Annihilability (BPUA) Assumption of Garg et al. [20].

In order to state the p -Bounded speedup hypothesis, we recall the following definition of Miles et al. [24].

Definition 5 (X-Max-2-SAT Solver). Consider a set $X \subseteq \{0, 1\}^\ell$. We say that an algorithm \mathcal{A} is an X-Max-2-SAT solver if it solves the Max-2-SAT problem restricted to inputs in X . Namely given a 2-CNF formula ϕ on ℓ variables, $\mathcal{A}(\phi) = 1$ iff $\exists x \in X$ that satisfies at least a $7/10$ fraction of ϕ 's clauses.

Assumption 1. (p -Bounded Speedup Hypothesis, introduced in [24]). Let $p : \mathbb{N} \rightarrow \mathbb{N}$. Then for any X-Max-2-SAT solver that has size $t(\ell)$, $|X| \leq p(\text{poly}(t(\ell)))$.

The assumption essentially states that the NP-complete problem Max-2-SAT is still hard even for restricted sets of variable assignments. This hardness is parameterized by p , and in its strongest form, p is taken to be a polynomial. In this form, the assumption states that no polynomial time algorithm can solve X-Max-2-SAT on an X of super-polynomial size. However, we can also take p to be $2^{\text{polylog}(n)}$ and obtain meaningful results.

We now state a non-uniform variant of the BPUA, but first we need the following definition from [20].

¹⁰ We set $k = 5$ so that the dimension of the random block diagonals added during encoding match the dimension of matrix branching programs obtained from Barrington's theorem.

Definition 6. A matrix branching program BP is L -bounded for $L \in \mathbb{N}$ if every intermediate value computed when evaluating BP on any input is at most L . In particular all of BP 's outputs and matrix entries are at most L .

Assumption 2. (Non-uniform variant of the BPUA assumption of [20]) Let $t = \text{poly}(\ell, \lambda)$ and let $\mathcal{X} \subseteq \{0, 1\}^\ell$ have $\text{poly}(\lambda)$ size and Q be a $\text{poly}(\lambda)$ -size $2^{o(\lambda)}$ -degree polynomial over \mathbb{Z} . Then for all ℓ , sufficiently large λ , and all primes $2^\lambda < p < 2^{\text{poly}(\lambda)}$, there exists a 2^λ -bounded dual-input matrix branching program $BP : \{0, 1\}^\ell \rightarrow [2^\lambda]$ of length t whose first input selection function (inp_1) iterates over the ℓ input bits t/ℓ times, such that $Q(\{BP(x)\}_{x \in \mathcal{X}}) \neq 0 \pmod{p}$.

Note that this statement is a very mild strengthening of the original BPUA assumption stated in [20]. Their assumption is required to hold for any Q of bounded degree generated by a polynomial-time algorithm, whereas our assumption must hold for any Q of polynomial size and bounded degree. However, we note that Garg et al. [20] justify their assumption by showing it is implied by the existence of PRFs in NC^1 secure against P/poly . With a minor tweak to their proof, we can show our non-uniform BPUA is also implied by the existence of PRFs in NC^1 secure against P/poly . We simply modify the non-uniform adversary used in [Theorem 2, [20]] to take the polynomial-size Q as advice.

Finally, we use the following definition in our security proof.

Definition 7 (Input-Respecting Polynomial). Given a branching program $\{\mathbf{M}_{i,b_1,b_2}\}_{i \in [h], b_1, b_2 \in \{0, 1\}}$ with input selection function $\text{inp} : [h] \rightarrow [\ell] \times [\ell]$, a polynomial p over the matrices (or elements of matrices) is input-respecting if no monomial involves two encodings $\{\mathbf{M}_{i,b_1^{(i)},b_2^{(i)}}\}, \{\mathbf{M}_{j,b_1^{(j)},b_2^{(j)}}\}$ (or entries of encodings) such that $\text{inp}(i)_1 = \text{inp}(j)_1$ and $b_1^{(i)} \neq b_1^{(j)}$ or $\text{inp}(i)_2 = \text{inp}(j)_2$ and $b_2^{(i)} \neq b_2^{(j)}$.

Theorem 2 (Main Theorem). Assuming the p -Bounded Speedup Hypothesis and the non-uniform BPUA Assumption (implied by the existence of PRFs in NC^1 secure against P/poly), our obfuscator is secure in the GGH15 Zeroizing Model.

Proof. It suffices to prove security in the GGH15 Annihilation Model with parameter 5 (since we set $k = 5$ in the obfuscation construction). Suppose an adversary \mathcal{A} wins in this model instantiated with our obfuscator. We argue that every successful zero-test polynomial submitted by \mathcal{A} is a linear combination of polynomially many branching program evaluations and thus that the existence of a Q used to win in the GGH15 Annihilation Model would violate Assumption 2. We know that every successful zero-test polynomial submitted by \mathcal{A} in this model is level-respecting, so by construction of straddling sets, we can conclude that every polynomial is input-respecting. A polynomial that is both edge-respecting (so each monomial contains exactly one branching program matrix from each layer) and input-respecting, is a linear combination of branching program evaluations. However, we have no bound on the number of terms in the linear combination. We now rely on the analysis techniques of Miles, Sahai, and Weiss [24] to show that each polynomial is in fact a linear combination of polynomially many branching program evaluations, assuming the p -Bounded Speedup Hypothesis.

Lemma 8. (adapted from [24]) Consider an adversary \mathcal{A} interacting with our obfuscation candidate in the GGH15 Annihilating Model. Assuming the p -Bounded Speedup Hypothesis, any edge-respecting and input-respecting polynomial submitted by \mathcal{A} is a linear combination of polynomially-many branching program evaluations.

Proof. Let q be an edge-respecting and input-respecting polynomial over handles to encodings $\{\widehat{\mathbf{C}}_{i,b_{i,1},b_{i,2}}\}$ submitted by \mathcal{A} (and thus can be represented as a polynomial size circuit c). We know that every monomial of q represents a branching program evaluation and thus can be associated with some input string $x \in \{0, 1\}^\ell$. Let X be the set of all such x associated with a monomial in q . We give an X -Max-2SAT solver of size $\text{poly}(|c|)$ which is sufficient to complete the proof. We are given a 2-CNF formula $\phi : \{0, 1\}^\ell \rightarrow \{0, 1\}$ with m clauses where $m \leq 4\ell^2$ without loss of generality. Let $\{y_i\}_{i \in [\ell]}$ denote the set of variables in ϕ . Fix a clause c in ϕ and let (i, j) be the input bits read by c . The input x associated with each monomial of q determines

an assignment to the variables y_i and y_j in ϕ . We modify q so that the degree of each monomial associated with a satisfying assignment to clause c is reduced by one. To do this, take any layer k reading (i, j) that has not been used before and in q , set every $\widehat{\mathbf{C}}_{k, b_1, b_2}$ to 1 except for the (b_1, b_2) pair that doesn't satisfy c . We can always pick a layer that wasn't used before because each pair is read at least $4\ell^2$ times by construction. In the case that $i = j$, we can take any unused layer k reading (i, i') for some $i' \neq i$ and set every $\widehat{\mathbf{C}}_{k, b_1, b_2}$ to 1 except for the (at most) two that don't satisfy c .

After doing this for each of the m clauses, we have that q contains a monomial of degree at most $t - 7m/10$ iff some $x \in \mathcal{X}$ satisfies $7m/10$ of ϕ 's clauses. Let $q^{(d)}$ be the homogeneous degree- d portion of q , and define $q' = \sum_{d=1}^{t-7m/10} q^{(d)}$ which can be computed in time $\text{poly}(|c|)$ and has size $\text{poly}(|c|)$ by Lemma 2.1 from [24]. Then $q' \not\equiv 0$ iff some $x \in \mathcal{X}$ satisfies $7m/10$ of ϕ 's clauses, so our goal will be to identity test q' . If we select a set of evaluation points α uniformly at random, the Schwartz-Zippel lemma says that except with low probability, $q'(\alpha) = 0$ if and only if $q' \equiv 0$. So by choosing many sets of points uniformly at random, we can significantly reduce the error probability. In particular, with a $\text{poly}(|c|)$ sized set of evaluation points, we can reduce the error to $< \frac{1}{2^{|c|}}$. Note that $2^{|c|}$ is an upper bound on the set of polynomials that q' was drawn from. Then by a union bound, the set of evaluation points we picked resulted in a correct identity testing algorithm for all possible q' simultaneously with probability strictly greater than $1 - \frac{1}{2^{|c|}}$. This implies the existence of some set of evaluation points which is perfectly correct on all q' simultaneously, so we can fix these points to give an X -Max-2SAT solver of size $\text{poly}(|c|)$. \square

With this lemma in hand, we inspect the Q submitted by \mathcal{A} that resulted in the model outputting “Win”. Notice that the $\{\widehat{\mathbf{C}}_i\}_i$ are in the shape of a dual-input branching program of width 5 (without the bookends), so by Lemma 8, every $\widehat{\mathbf{v}} \cdot p_u(\{\widehat{\mathbf{C}}_i\}_i) \cdot \widehat{\mathbf{w}}^\top$ is actually a linear combination of polynomially many honest branching program evaluations. Since there are only polynomially many p_u 's (since Q is implemented with a polynomial size circuit), and since Q is identically zero over these evaluations, Q contradicts Assumption 2, and we can conclude that \mathcal{A} could not have won in the GGH15 Annihilation model and thus in the GGH15 Zeroizing Model except with negligible probability. \square

Remark. If we instead take $p = 2^{\text{poly}(\log(n))}$ in Assumption 1, we can conclude that every input-respecting polynomial submitted by \mathcal{A} is a linear combination of quasi-polynomially many branching program evaluations. To argue security via BPUA, we can instead assume a PRF that is sub-exponentially secure against P/poly. The reason we consider a parameterized version of Assumption 1 is that, as mentioned in [24], this assumption with $p = \text{poly}(n)$ is similar in spirit to the Bounded Speedup Hypothesis of [35] (defined relative to 3-SAT) which has subsequently been shown to be false. The assumption we use here however is relative to a different NP-complete problem and, as noted in [24], can be reduced to other NP-complete problems as well.

6 Acknowledgements

We thank Yilei Chen for helpful discussions. We also thank the anonymous TCC reviewers for providing insightful feedback. 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., Raykova, M., Sahai, A., Waters, B.: Candidate indistinguishability obfuscation and functional encryption for all circuits. (2013) 40–49
3. Garg, S., Gentry, C., Halevi, S.: Candidate multilinear maps from ideal lattices. (2013) 1–17

4. Coron, J.S., Lepoint, T., Tibouchi, M.: Practical multilinear maps over the integers. (2013) 476–493
5. Gentry, C., Gorbunov, S., Halevi, S.: Graph-induced multilinear maps from lattices. (2015) 498–527
6. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. (2005) 84–93
7. Wichs, D., Zeldelis, G.: Obfuscating compute-and-compare programs under LWE. (2017) 600–611
8. Goyal, R., Koppula, V., Waters, B.: Lockable obfuscation. (2017) 612–621
9. Chen, Y., Vaikuntanathan, V., Wee, H.: GGH15 beyond permutation branching programs: Proofs, attacks, and candidates. (2018) 577–607
10. Canetti, R., Chen, Y.: Constraint-hiding constrained PRFs for NC¹ from LWE. (2017) 446–476
11. Goyal, R., Koppula, V., Waters, B.: Separating semantic and circular security for symmetric-key bit encryption from the learning with errors assumption. (2017) 528–557
12. Shor, P.W.: Algorithms for quantum computation: Discrete logarithms and factoring. (1994) 124–134
13. 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
14. Cramer, R., Ducas, L., Peikert, C., Regev, O.: Recovering short generators of principal ideals in cyclotomic rings. (2016) 559–585
15. Biasse, J.F., Song, F.: Efficient quantum algorithms for computing class groups and solving the principal ideal problem in arbitrary degree number fields. (2016) 893–902
16. Pellet-Mary, A.: Quantum attacks against indistinguishability obfuscators proved secure in the weak multilinear map model. (2018) 153–183
17. Cheon, J.H., Han, K., Lee, C., Ryu, H., Stehlé, D.: Cryptanalysis of the multilinear map over the integers. (2015) 3–12
18. Coron, J.S., Lee, M.S., Lepoint, T., Tibouchi, M.: Cryptanalysis of GGH15 multilinear maps. (2016) 607–628
19. Chen, Y., Gentry, C., Halevi, S.: Cryptanalyses of candidate branching program obfuscators. (2017) 278–307
20. Garg, S., Miles, E., Mukherjee, P., Sahai, A., Srinivasan, A., Zhandry, M.: Secure obfuscation in a weak multilinear map model. (2016) 241–268
21. Ma, F., Zhandry, M.: The mmap strikes back: Obfuscation and new multilinear maps immune to CLT13 zeroizing attacks. In: TCC 2018. (2018)
22. Barak, B., Garg, S., Kalai, Y.T., Paneth, O., Sahai, A.: Protecting obfuscation against algebraic attacks. (2014) 221–238
23. Halevi, S.: Graded encoding, variations on a scheme. Cryptology ePrint Archive, Report 2015/866 (2015) <http://eprint.iacr.org/2015/866>.
24. Miles, E., Sahai, A., Weiss, M.: Protecting obfuscation against arithmetic attacks. Cryptology ePrint Archive, Report 2014/878 (2014) <http://eprint.iacr.org/2014/878>.
25. Badrinarayanan, S., Miles, E., Sahai, A., Zhandry, M.: Post-zeroizing obfuscation: New mathematical tools, and the case of evasive circuits. (2016) 764–791
26. 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
27. Goldwasser, S., Kalai, Y.T.: Cryptographic assumptions: A position paper. (2016) 505–522
28. Halevi, S., Halevi, T., Shoup, V., Stephens-Davidowitz, N.: Implementing BP-obfuscation using graph-induced encoding. (2017) 783–798
29. Boneh, D., Lewi, K., Montgomery, H.W., Raghunathan, A.: Key homomorphic PRFs and their applications. (2013) 410–428
30. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: Proceedings of the fortieth annual ACM symposium on Theory of computing, ACM (2008) 197–206
31. Garg, S., Miles, E., Mukherjee, P., Sahai, A., Srinivasan, A., Zhandry, M.: Secure obfuscation in a weak multilinear map model. Cryptology ePrint Archive, Report 2016/817 (2016) <http://eprint.iacr.org/2016/817>.
32. Barrington, D.A.M.: Bounded-width polynomial-size branching programs recognize exactly those languages in NC¹. (1986) 1–5
33. Lenstra, A.K., Lenstra, H.W., Lovász, L.: Factoring polynomials with rational coefficients. Mathematische Annalen **261**(4) (1982) 515–534
34. Menezes, A.J., Okamoto, T., Vanstone, S.A.: Reducing elliptic curve logarithms to logarithms in a finite field. IEEE Transactions on Information Theory **39**(5) (Sept 1993) 1639–1646
35. Brakerski, Z., Rothblum, G.N.: Virtual black-box obfuscation for all circuits via generic graded encoding. (2014) 1–25

A Weaker GGH15 Zeroizing Models

We describe two separate relaxations of our GGH15 Zeroizing Model, designed to allow the adversary more freedom in how it can potentially obtain successful zero-test results. We prove that when instantiated with γ -GGH15-AL, security in the GGH15 Zeroizing Model implies security in each of these models (in one case we actually require a small tweak to γ -GGH15-AL).

A.1 Edge-Relaxed GGH15 Zeroizing Model

First, we relax the condition that the adversary must submit edge-respecting polynomials over encodings and we refer to the resulting model as the Edge-Relaxed GGH15 Zeroizing Model. The only difference lies in the description of zero-testing - when the adversary submits a polynomial p over encodings, the model no longer checks that p is edge-respecting and instead checks that p has degree $2^{o(\lambda)}$ over encodings. We now tweak γ -GGH15-AL by adding “edge scalars” in the same manner that we added level scalars in Section 4.3. The KeyGen and Enc procedures change slightly and we describe the differences below.

- γ -GGH15-AL.KeyGen: For each $i \in [d-1]$, choose a random invertible scalar $r_i \in \mathbb{Z}_q$, the new right bookend will be

$$\mathbf{w} = \left(\prod_{i \in [d-1]} r_i \right) \left(\prod_{\ell \in [\kappa]} z_\ell \right) \cdot \mathbf{R}_d^{-1} \cdot \begin{bmatrix} \mathbf{D}^* \cdot \mathbf{w}'^\top \\ \mathbf{b}_w^\top \end{bmatrix}$$

and we add $\{r_i\}_{i \in [d-1]}$ to \mathbf{sp} .

- γ -GGH15-AL.Enc: In order to produce an encoding \mathbf{C} on the path $i \rightsquigarrow j$ at level L , we compute

$$\mathbf{C} = \left(\prod_{k \in \{i, \dots, j-1\}} r_k \right)^{-1} \left(\prod_{\ell \in L} z_\ell \right)^{-1} \cdot \mathbf{R}_i^{-1} \cdot \begin{bmatrix} \mathbf{D} \\ \mathbf{B} \end{bmatrix} \cdot \mathbf{R}_j.$$

This allows us to prove that the Edge-Relaxed GGH15 Zeroizing Model is actually equivalent to the GGH15 Zeroizing Model when instantiated with our construction.

Lemma 9. *An adversary \mathcal{A} that wins in the Edge-Relaxed GGH15 Zeroizing Model instantiated with γ -GGH15-AL (with edge scalars) implies an adversary \mathcal{A}' that wins in the GGH15 Zeroizing Model instantiated with γ -GGH15-AL (with edge scalars).*

Proof. We show that \mathcal{A} already satisfies the win condition of the GGH15 Zeroizing Model, so $\mathcal{A}' = \mathcal{A}$. We first argue that every monomial in a successful zero-test polynomial submitted by \mathcal{A} must be a permutation of an edge-respecting monomial. This follows from the analysis used in proving lemma 6 by considering the set of $\{r_i\}$ as formal variables. We do have to bound the degree of the polynomial in this case to $2^{o(\lambda)}$ in order to apply the Schwartz-Zippel lemma to conclude the above holds with $1 - \text{negl}(\lambda)$ probability.

Next, we show that each monomial must actually be edge-respecting by fixing the correct ordering. We use the Kilian randomization of encodings and a variant of a lemma by Ma and Zhandry [21] in order to show this. In order to make use of this lemma, we have to supply several definitions from [21].

We begin by considering a collection of n columns of matrices, where each column may contain an arbitrary polynomial number of matrices. Denote the j -th matrix in column i as $\mathbf{A}_{i,j}$. 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. Further suppose that multiplying one matrix from each column results in a scalar. Square matrices \mathbf{R}_i are chosen at random. Then matrix $\mathbf{A}_{i,j}$ is left-multiplied by \mathbf{R}_i^{-1} and right-multiplied by \mathbf{R}_{i+1} to form $\tilde{\mathbf{A}}_{i,j}$. We also consider a set S of n -tuples of indices that define a set of “allowable” iterated matrix products.

Definition 8 (Allowable Polynomials [21]). *Given $\{\tilde{\mathbf{A}}_{i,j}\}$ and S as described, an allowable polynomial over S is a 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 .*

Definition 9 (Allowable Matrix Products [21]). An allowable matrix product over S is the iterated matrix product corresponding to some tuple $t \in S$.

[21] notes that allowable matrix products are an example of allowable polynomials.

We consider each entry of the $\{\mathbf{A}_{i,j}\}$ matrices described above as a polynomial over a set of variables X .

Definition 10 (Linear Independence [21]). Let $X = \{X_1, \dots, X_m\}$ denote a set of variables. For a set of vectors $V = \{\mathbf{v}_1, \dots, \mathbf{v}_\ell\}$ whose entries are polynomials over X , we say that V is linearly independent if there is no sequence of constants a_1, \dots, a_ℓ , not all of which are 0, such that $\sum_{i \in [\ell]} a_i \mathbf{v}_i$ is the identically 0 vector.

Definition 11 (Left / Right Non-Shortcutting [21]). A collection of matrices $\{\mathbf{A}_{i,j}\}$ of polynomials over the variables X along with a set S of valid matrix products satisfy left non-shortcutting if the following holds. For each member of S , consider multiplying every corresponding matrix except for the rightmost matrix and interpret the resulting matrices as vectors of polynomials over the variables X . These vectors must be linearly independent (in the sense defined above). Right non-shortcutting is defined analogously.

With these definitions in hand, we are now able to state a variant of the Lemma 2 from [21]. We justify this variant after completing the proof.

Lemma 10 (Modification of Lemma 2 from [21]). Suppose a collection of matrices $\{\mathbf{A}_{i,j}\}$ of polynomials over the variables X along with the set S of valid products satisfy left and right non-shortcutting. Then any allowable polynomial over the $\tilde{\mathbf{A}}_{i,j}$ matrices that is identically a constant as a rational function over the \mathbf{R} variables can be written as a linear combination of allowable matrix products over S .

Returning to our model, we set up columns of matrices that fit the structure of the $\{\tilde{\mathbf{A}}_{i,j}\}$ described above and allow us to invoke Lemma 10. The columns range from column 0 up to column d . For $1 \leq i \leq d$, place all matrices \mathbf{C} encoded on a path starting at node i in column i . \mathbf{v} will be the only ‘matrix’ in column 0, and \mathbf{w} will be the only matrix in column d . We also add a Kilian-randomized identity matrix to each column. Thus the entire set of encodings \mathbf{C} , the bookends \mathbf{v} and \mathbf{w} , and the identity matrices, correspond to the collection $\{\tilde{\mathbf{A}}_{i,j}\}$ of matrices from the lemma (note that we can re-define the Kilian randomization on an encoding on path $i \rightsquigarrow j$ to be right-multiplied by \mathbf{R}_{i+1} instead of by \mathbf{R}_{j+1} and then take a product with Kilian-randomized identity matrices at columns $i+1, \dots, j$ to recover the original encoding).

Re-express each entry of each encoding \mathbf{C} and the bookends \mathbf{v} and \mathbf{w} as polynomials over the set of variables $\{\hat{b}_i\}$ added as random block diagonals (these correspond to the variables X in the lemma statement). Many entries will be constant (in particular entries of the \mathbf{D} matrices), but importantly the formal variables appearing in any matrix will be distinct from the ones appearing in the other matrices.

The first part of this proof showed that due to the random edge scalars, each polynomial that \mathcal{A} submits (viewed now as a polynomial over *entries* of encoding matrices) must be an allowable polynomial over the set S , where S consists of all possible $(d+1)$ -tuples of indices subject to the constraint that whenever the tuple includes an element from some matrix \mathbf{C} encoded at edge $i \rightsquigarrow j$, the elements from columns $i+1, \dots, j$ must come from the identity matrix. We note that the block diagonal variables $\{\hat{b}_i\}$ ensure that this collection of matrices along with S satisfies left and right non-shortcutting. We are then able to apply Lemma 10 to conclude that the polynomial \mathcal{A} submitted must in fact be a linear combination of *iterated* matrix products over S , which ensures that \mathcal{A} submitted a polynomial (now viewed over encodings rather than over elements of encodings) where every monomial is an iterated product of encoding matrices that form a source-to-sink path. This completes the proof of Lemma 9.

□

Justification of Lemma 10

First, we note that if we replace the \mathbf{R}_i^{-1} 's with $\mathbf{R}_i^{\text{adj}}$'s in the construction of the $\tilde{\mathbf{A}}_{i,j}$ matrices, then Lemma 10 from above is equivalent to the following, since the determinants will cancel out:

Lemma 11. Suppose a collection of matrices $\{\mathbf{A}_{i,j}\}$ of polynomials over the variables X along with the set S of valid products satisfies left and right non-shortcutting. Then any allowable polynomial over the $\tilde{\mathbf{A}}_{i,j}$ matrices that is identically $C \prod_{i=1}^{n-1} \det(\mathbf{R}_i)$ for some constant C as a polynomial over the \mathbf{R} variables can be written as a linear combination of allowable matrix products over S .

Now that we are dealing with adjoint matrices, we can mirror the proof of lemma 2 from [21]. The proof begins by expanding out an arbitrary allowable polynomial over the \mathbf{R} and $\mathbf{A}_{i,j}$ variables. The first difference in our setting is when the proof examines the types of products of entries in \mathbf{R}_1 that are possible. This product being ‘well-formed’ means it is a permutation monomial and ‘mal-formed’ means it is a non-permutation monomial. Here, well-formed products can actually have a non-zero coefficient, namely $C \prod_{i=2}^{n-1} \det(\mathbf{R}_i)$. Importantly, all mal-formed products still must be zero. The inductive step only makes use of the fact that mal-formed products are zero to conclude that any monomial that doesn’t result from proper multiplication of two matrices in the first two columns is zeroed out. This is enough to show the result.

A.2 Matrix-Relaxed GGH15 Zeroizing Model

We now turn to describing a relaxation of the GGH15 Zeroizing Model in which we no longer give out handles to the entire matrix of each encoding and instead give out handles to each element of each matrix. We appeal to the Kilian-randomization and the above lemma adapted from [21] to immediately conclude equivalence with the GGH15 Zeroizing Model when instantiated with γ -GGH15-AL (note that we no longer need edge scalars in this setting). We describe the modifications we make to the model below.

Initialize Elements. After producing an encoding \mathbf{C}_i , the model \mathcal{M} releases a handle to each element of the set $\{\hat{C}_{i,j,k}\}$, where $C_{i,j,k}$ is the (j, k) -th element of encoding \mathbf{C}_i .

Zero-Test. The adversary now submits a matrix \mathbf{M} of polynomials for zero-testing, where $M_{s,t} = p_{s,t}(\{\hat{C}_{i,j,k}\}_{i,j,k})$ is a polynomial over handles to elements of encodings. First, \mathcal{M} checks that each $p_{s,t}$ is edge-respecting (each monomial consists of exactly one entry plucked from each matrix in a source-to-sink path). Now letting $M(\{C_{i,j,k}\}_{i,j,k})$ denote the matrix that results from plugging in encoding entries, the model \mathcal{M} obtains $(T, b) \leftarrow \text{G.ZeroTest(pp, } M(\{C_{i,j,k}\}_{i,j,k}))$ and proceeds as in the GGH15 Zeroizing Model.

Lemma 12. An adversary \mathcal{A} that wins in the Matrix-Relaxed GGH15 Zeroizing Model instantiated with γ -GGH15-AL implies an adversary \mathcal{A}' that wins in the GGH15 Zeroizing Model instantiated with γ -GGH15-AL.

Proof. We set up columns required to invoke Lemma 10 in the same manner as in the previous proof. The edge-respecting condition gives us that each polynomial submitted by \mathcal{A} is an allowable polynomial over the same set S , which allows us to immediately conclude that it is a linear combination of iterated matrix products, so $\mathcal{A} = \mathcal{A}'$. \square

Remark. It would be ideal to combine both relaxations into just one model and then to prove a corresponding model conversion lemma. However, this conversion would seem to require each individual entry of each encoding to have large independent entropy. This appears tricky to argue since columns of (the non-block diagonal part of) each encoding are drawn together in a random lattice coset. Thus it is clear that there is some independent entropy between different columns of encodings but not as clear that there is enough independent entropy among elements of the same column.

B Settings of the GGH15 Zeroizing Attacks

The known zeroizing attacks on GGH15 are mounted on slightly different variants of the scheme, with different sets of encoded plaintexts and underlying graph structures. In this section, we review the settings of the three attacks and rewrite them in more unified notation.

B.1 CLLT16 Attack on GGH15 Key Exchange

We recall the GGH15 multiparty key agreement protocol, which operates over the commutative variant of GGH15 from [5]. We consider the protocol with 3 users, and note that both the protocol and the attack can be easily generalized to $k > 3$ users. For each user $i \in \{1, 2, 3\}$, we construct a directed path as in Figure 1 with $k + 1 = 4$ nodes, namely $v_{i,1} \rightarrow v_{i,2} \rightarrow v_{i,3} \rightarrow v_{i,4}$, and associate each of the nodes with a row vector $\mathbf{A}_{v_{i,1}}, \dots, \mathbf{A}_{v_{i,4}}$ (we will refer to these vectors as matrices for consistency of notation between the attack descriptions). The paths of different users share the same end-point $v_0 = v_{i,4}$ for all $i \in \{1, 2, 3\}$. Consequently, $\mathbf{A}_{v_0} = \mathbf{A}_{v_{i,4}}$ for all $i \in \{1, 2, 3\}$.

The protocol produces a number of public encodings for each edge, corresponding to small secret components $s_{i,l}$ ¹¹ for $1 \leq i \leq 3$ and $1 \leq l \leq N$, for large enough N . These secret components are generated randomly, and are encoded on edge $i' = i + j - 1$ for each path j , denoted as $\mathbf{C}_{j,i',l}$. For our case with 3 users, we have the following encodings for User 1:

$$\begin{aligned}\mathbf{A}_{v_{1,1}} \cdot \mathbf{C}_{1,1,l} &= s_{1,l} \cdot \mathbf{A}_{v_{1,2}} + \mathbf{E}_{1,1,l} \pmod{q} \\ \mathbf{A}_{v_{2,2}} \cdot \mathbf{C}_{2,2,l} &= s_{1,l} \cdot \mathbf{A}_{v_{2,3}} + \mathbf{E}_{2,2,l} \pmod{q} \\ \mathbf{A}_{v_{3,3}} \cdot \mathbf{C}_{3,3,l} &= s_{1,l} \cdot \mathbf{A}_{v_0} + \mathbf{E}_{3,3,l} \pmod{q}\end{aligned}$$

where \mathbf{E} denotes the error matrices (vectors). Note that there is no pre-processing step in this scheme as the plaintext $s_{i,l}$ is directly used as the LWE secret.

The original description of the Coron et al. attack does not directly construct top-level encodings of zero in a strict sense. Instead, it zero-tests two source-to-sink encodings of the same plaintext (relative to different sources) and subtracts the result. By modifying the graph to have a single “super” source node with directed edges to each of the original source nodes and having each of these edges encode a plaintext value of 1, the original procedure of computing the difference between two same encodings relative to two different paths becomes computing the difference between two same encodings relative to the same source-to-sink path¹², which is essentially computing a single top-level encoding of zero.

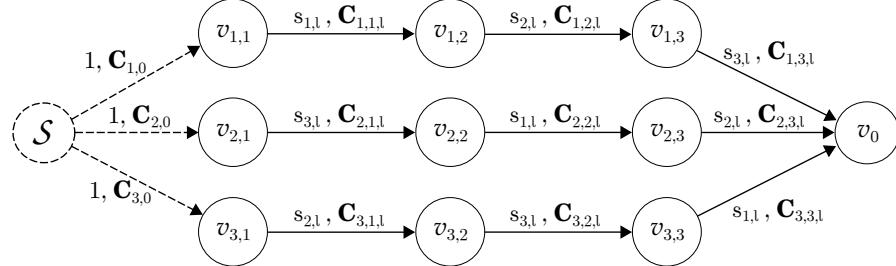


Fig. 1. The GGH15 graph structure for a 3-party key agreement protocol with the “super” source node \mathcal{S} added.

Now consider the following polynomial:

$$\mathbf{C}_{2,0} \cdot \mathbf{C}_{2,1,1} \cdot \mathbf{C}_{2,2,j} \cdot \mathbf{C}_{2,3,k} - \mathbf{C}_{3,0} \cdot \mathbf{C}_{3,1,k} \cdot \mathbf{C}_{3,2,1} \cdot \mathbf{C}_{3,3,j}$$

for some j, k that satisfies $1 \leq j, k \leq N$. Notice that $\mathbf{C}_{2,0} \cdot \mathbf{C}_{2,1,1} \cdot \mathbf{C}_{2,2,j} \cdot \mathbf{C}_{2,3,k}$ encodes $1 \cdot s_{3,1} \cdot s_{1,j} \cdot s_{2,k}$ on the path for User 2, and $\mathbf{C}_{3,0} \cdot \mathbf{C}_{3,1,k} \cdot \mathbf{C}_{3,2,1} \cdot \mathbf{C}_{3,3,j}$ encodes $1 \cdot s_{2,k} \cdot s_{3,1} \cdot s_{1,j}$ on the path for User 3. By

¹¹ These secret components are denoted as $t_{i,l}$ in the original paper.

¹² Although these two source-to-sink paths have different intermediate nodes, they are considered the same path in GGH15 as long as they share the same start and end nodes.

commutativity of scalar multiplication, these are encodings of the same plaintext element. Since they're both encoded relative to the same source-to-sink path, the difference between these two encodings constitutes a top-level encoding of zero.

B.2 CGH17 Attack on GGHRSW Obfuscation

To construct the GGHRSW obfuscation candidate over GGH15, we use the original, non-commutative GGH15 scheme. We consider a graph with two parallel chains leading to the same sink, corresponding to the functional branch and the dummy branch of the GGHRSW obfuscator, as illustrated in Figure 2. Each chain consists of $h + 2$ nodes, where h is the number of branching program layers. Namely we have $v_1 \rightarrow \dots \rightarrow v_{h+2}$ for the functional branch chain and $v'_1 \rightarrow \dots \rightarrow v'_{h+1} \rightarrow v_{h+2}$ for the dummy branch chain. The construction encodes the matrices $\{\mathbf{M}_{i,b}\}_{i \in [h], b \in \{0,1\}}$ of a branching program by first applying pre-processing in the form of adding block diagonals, Kilian randomization, and bundling scalars to produce the set $\{\mathbf{S}_{i,b}\}_{i \in [h], b \in \{0,1\}}$. For all $i \in [h]$, it then encodes $\mathbf{S}_{i,b}$ for $b \in \{0,1\}$ on edge $v_i \rightarrow v_{i+1}$, denoted as $\mathbf{C}_{i,b}$. Similarly, we have $\mathbf{C}'_{i,b}$ encoding the pre-processed $\mathbf{S}'_{i,b}$ matrices of a branching program consisting of all identity matrices.

Recall that in the GGHRSW construction, the functional branch uses two bookend vectors, denoted as J and L , where J is a row vector of dimension m and L is a column vector of dimension m . Similarly, we have J' and L' for the dummy branch. However, the plaintext space of GGH15 only contains matrices of size $m \times m$. This is handled by choosing these GGHRSW bookends as matrices instead of vectors. Namely, we choose $\mathbf{S}_0, \mathbf{S}_{h+1}, \mathbf{S}'_0, \mathbf{S}'_{h+1}$ ¹³ bookend matrices by repeating the rows / columns of J, L, J', L' . We encode \mathbf{S}_{h+1} and \mathbf{S}'_{h+1} on edges $v_{h+1} \rightarrow v_{h+2}$ and $v'_{h+1} \rightarrow v_{h+2}$ as \mathbf{C}_{h+1} and \mathbf{C}'_{h+1} ¹⁴ respectively.

As in the CLLT16 attack, we apply the same technique of adding a “super” source node with directed edges to the original source nodes of the two chains. Namely, we add a new node \mathcal{S} and two direct edges $\mathcal{S} \rightarrow v_1$ and $\mathcal{S} \rightarrow v'_1$. Additionally, we encode \mathbf{S}_0 and \mathbf{S}'_0 on $\mathcal{S} \rightarrow v_1$ and $\mathcal{S} \rightarrow v'_1$ as \mathbf{C}_0 and \mathbf{C}'_0 .

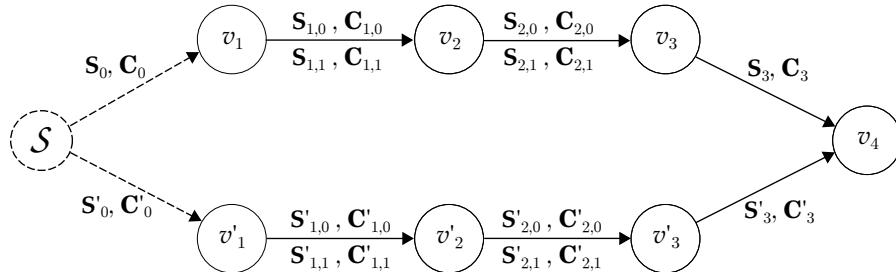


Fig. 2. The GGH15 graph structure for a GGHRSW obfuscator with 2 branching program layers augmented with a “super” source node \mathcal{S} .

For a branching program input $u \in [\ell]$, denote u_α for $\alpha \in [\ell]$ as the α -th input bit. Additionally, define the input mapping function $\text{inp} : [h] \rightarrow [\ell]$ which maps a branching program layer to the input bit that should be read at that layer. The result of a branching program evaluation thus takes the form of

$$\mathbf{S}_0 \cdot \prod_{i \in [h]} \mathbf{S}_{i, u_{\text{inp}(i)}} \cdot \mathbf{S}_{h+1} - \mathbf{S}'_0 \cdot \prod_{i \in [h]} \mathbf{S}'_{i, u_{\text{inp}(i)}} \cdot \mathbf{S}'_{h+1}$$

¹³ In the original CGH17 paper, these are denoted as $\mathbf{J}, \mathbf{L}, \mathbf{J}',$ and \mathbf{L}' .

¹⁴ Originally denoted as $\tilde{\mathbf{L}}$ and $\tilde{\mathbf{L}}'$.

where the first monomial corresponds to the branching program evaluated on the functional branch, and the second corresponds to that on the dummy branch. Hence, the encoding of the evaluation result has the form

$$\mathbf{C}_0 \cdot \prod_{i \in [h]} \mathbf{C}_{i, u_{\text{inp}(i)}} \cdot \mathbf{C}_{h+1} - \mathbf{C}'_0 \cdot \prod_{i \in [h]} \mathbf{C}'_{i, u_{\text{inp}(i)}} \cdot \mathbf{C}'_{h+1}.$$

It is easy to verify that this corresponds to a top-level encoding, since both of the monomials are encoded relative to the same sink-to-source path.

Looking ahead, the CGH17 attack also requires that the branching program has an input partition $[h] = \mathcal{X} \parallel \mathcal{Z}$ which means that there are sufficiently many input bits which only control layers in \mathcal{X} and sufficiently many input bits which only control layers in \mathcal{Z} . Note that in input u , the bits that control layers in \mathcal{X} do not necessarily come before ones that control layers in \mathcal{Z} . Let the input be denoted as $u^{(j,k)} = x^{(j)}z^{(k)} \in \{0, 1\}^\ell$ and let $u_\alpha^{(j,k)}$ denote the α -th input bit. Then the encoding of a branching program evaluation will be

$$\mathbf{C}_0 \cdot \prod_{i \in \mathcal{X}} \mathbf{C}_{i, u_{\text{inp}(i)}^{(j,k)}} \cdot \prod_{i \in \mathcal{Z}} \mathbf{C}_{i, u_{\text{inp}(i)}^{(j,k)}} \cdot \mathbf{C}_{h+1} - \mathbf{C}'_0 \cdot \prod_{i \in \mathcal{X}} \mathbf{C}'_{i, u_{\text{inp}(i)}^{(j,k)}} \cdot \prod_{i \in \mathcal{Z}} \mathbf{C}'_{i, u_{\text{inp}(i)}^{(j,k)}} \cdot \mathbf{C}'_{h+1}.$$

B.3 CVW18 Attack on Obfuscation with Safeguards

The CVW18 attack targets a variant of the original GGHRSSW obfuscator, where many of the known safeguard mechanisms are applied. As illustrated in Figure 3, the underlying graph structure for this scheme is a simple path graph with $h + 1$ vertices, where h is the number of branching program layers. Namely, we have a graph $v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_{h+1}$. Branching program matrices $\{\mathbf{M}_{i,b}\}_{i \in [h], b \in \{0,1\}}$ are tensored with random matrices, appended with random block diagonals, and Kilian randomized to produce the pre-encodings $\{\mathbf{S}_{i,b}\}_{i \in [h], b \in \{0,1\}}$ ¹⁵ for $b \in \{0, 1\}$. These matrices are then encoded as $\{\mathbf{C}_{i,b}\}_{i \in [h], b \in \{0,1\}}$. Thus, the result of a branching program evaluation is simply $\prod_{i \in [h]} \mathbf{S}_{i, u_{\text{inp}(i)}}$, which is encoded as $\prod_{i \in [h]} \mathbf{C}_{i, u_{\text{inp}(i)}}$.

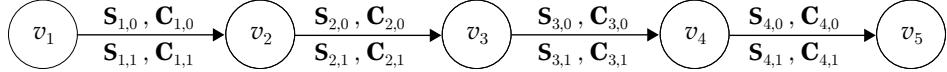


Fig. 3. The GGH15 graph structure for a GGHRSSW obfuscator variant with all the safeguards for a 4-layer branching program.

Like the CGH17 attack, this attack requires an input-partitioning on the branching program. They are slightly more general in that they require a c -input partition for constant c rather than just a 2-input partition as described above. However, larger values of c do not significantly alter the attack and we restrict attention to 2-input partitions here, noting that the extensions fit in our model in an analogous way. In the case of a 2-input partition $[h] = \mathcal{X} \parallel \mathcal{Z}$, the encoding of a branching program evaluation becomes

$$\prod_{i \in \mathcal{X}} \mathbf{C}_{i, u_{\text{inp}(i)}^{(j,k)}} \cdot \prod_{i \in \mathcal{Z}} \mathbf{C}_{i, u_{\text{inp}(i)}^{(j,k)}}.$$

C Proof of Theorem 1

We prove Theorem 1, which extends the semantic security proof (Theorem 5.7 of [9]) of the γ -GGH15 encodings of Chen, Vaikuntanathan, and Wee [9] to general directed acyclic graphs.

¹⁵ Originally denoted as $\hat{\mathbf{S}}_{i,b}$.

Proof. We briefly show how to modify the original proof of [9] to handle this general setting. To keep the length of this section reasonable, we only highlight the parts of our proof that differ from theirs. We refer the reader to [9] for the original proof.

- We define Distributions 1.i for $i \in \{d+2, d+1, \dots, 2\}$ analogously to how they are defined in [9]. In their setting, to go from Distribution 1.($i+1$) to Distribution 1. i , Chen et al. change how $\mathbf{A}_{i-1}, \mathbf{D}_{i,0}, \mathbf{D}_{i,1}$ are sampled. More generally, this step re-samples all encodings that require a trapdoor for \mathbf{A}_{i-1} , which are just the two encodings $\mathbf{D}_{i,0}, \mathbf{D}_{i,1}$ on the edge $(i-1, i)$. The only difference in our setting is that the encodings on edges starting at $i-1$ are ones of the form $\mathbf{D}_{(i-1,i'),j}$ for any j and any $i' > i$. To go between distributions we change how all such $\mathbf{D}_{(i-1,i'),j}$'s are sampled.
- We can extend the proof of Lemma 5.10 in [9] to our setting. In their proof, Chen et al. construct an intermediate distribution 1. i^* and show it is indistinguishable from 1. i . This part of the proof is completely unaffected by changing the number of \mathbf{D} encodings. The second part of their proof requires showing 1. i^* is indistinguishable from 1.($i+1$). This follows from showing that a distinguisher would be able to distinguish two LWE samples with common secret $\underline{\mathbf{A}}_i$. We simply extend this to the case where we have some samples with common secret $\underline{\mathbf{A}}_i$, other samples with common secret $\underline{\mathbf{A}}_{i+1}$, etc. To do this we step through a hybrid for each possible value of i' , in which we (roughly) replace all $\mathbf{S}_{(i-1,i'),j} \mathbf{A}_{(i-1,i'),j} + \mathbf{E}_{(i-1,i'),j}$ for some fixed i' with independent uniform samples. Indistinguishability of these hybrid steps is proved with the same LWE distinguisher constructed in the proof of Lemma 5.10 in [9].
- We define Distributions 2. i for $i = 2, 3, \dots, d+1$ to be the same as in [9], except instead of changing how $\mathbf{D}_{i,0}, \mathbf{D}_{i,1}$ are sampled, we change all encodings of the form $\mathbf{D}_{(i-1,i'),j}$ for any $i' > i-1$ and for any j .
- In the proof of Lemma 5.11, we simply change their matrices $[\mathbf{M}_{j,0} \mathbf{A}_j \mid \mathbf{M}_{j,1} \bar{\mathbf{A}}_j]$ to be a concatenation of all matrices of the form $\mathbf{M}_{(i-1,i'),j} \bar{\mathbf{A}}_{i'}$. The rest of the proof generalizes naturally.
- We note that the original proof assumes $\text{LWE}_{m, 2m, q, U(\mathbb{Z}_q), D_{\mathbb{Z}, \sigma}, D_{\mathbb{Z}, \sigma}}$, where the $2m$ parameter is due to there being 2 encodings relative to each vertex. To extend the proof to the more general setting, we need to replace $2m$ with $|e|m$, where $|e|$ denotes the maximum number of encodings encoded relative to any vertex in the DAG.

□

D Asymmetric Levels from Level Gadgets

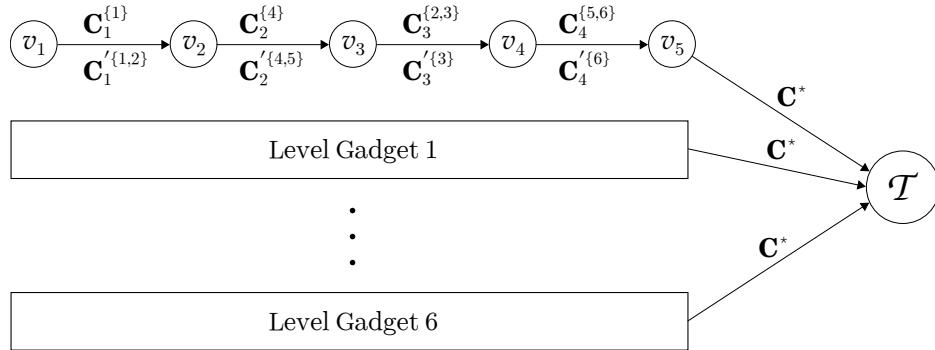
Here we present an alternative way to construct a graph-induced encoding scheme with asymmetric levels using “level gadgets”. We make use of an observation from [7] that GGH15 can be tweaked to support *simultaneous* encodings of a plaintext \mathbf{M} . Namely, \mathbf{M} can be encoded on the set of edges $\{i_1 \rightsquigarrow j_1, \dots, i_k \rightsquigarrow j_k\}$ by sampling \mathbf{D} such that

$$\begin{bmatrix} \mathbf{A}_{i_1} \\ \vdots \\ \mathbf{A}_{i_k} \end{bmatrix} \mathbf{D} = \begin{bmatrix} \mathbf{M} \cdot \mathbf{A}_{j_1} + \mathbf{E}_1 \\ \vdots \\ \mathbf{M} \cdot \mathbf{A}_{j_k} + \mathbf{E}_k \end{bmatrix}$$

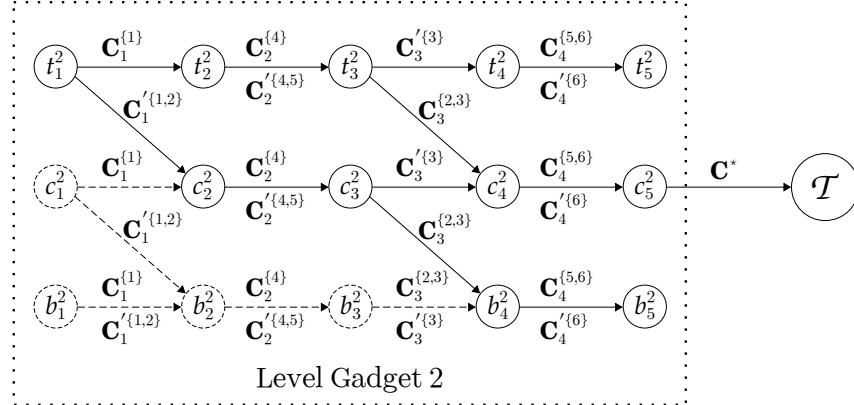
We will make use of this observation to build asymmetric levels in the following way. To support a level set of size κ , we first produce 3κ copies of the input graph and arrange them in groups of 3 (level gadgets). We refer to the original graph as the “main branch”. We associate one source with each level gadget (so we now have $\kappa+1$ distinct sources) and we create a new node that acts as a “super” sink. Our goal is to encode plaintexts simultaneously on the main branch and the level gadgets in such a way that top-level products of plaintexts are level-respecting if and only if they are simultaneously encoded from each source to the super sink. We accomplish this by viewing each level gadget as three layers - top, center, and bottom (t, c, b) - where each is a copy of the main branch (see figure 1). The source we associate with each level gadget is the first node of the top layer, and for each gadget, we form an edge from the last node of the center layer

to the super sink. In order to zero-test an encoding successfully, it at least must be simultaneously encoded relative to each of the source nodes.

Now say that we want to encode a plaintext \mathbf{M} on edge $i \rightsquigarrow j$ at level ℓ (more generally this can be a set of levels). We encode it simultaneously on $3\kappa + 1$ edges, the first being the edge $i \rightsquigarrow j$ on the main branch. On level gadget ℓ , we explicitly move down one layer, meaning we encode \mathbf{M} on edge $t_i^\ell \rightsquigarrow c_j^\ell$ and edge $c_i^\ell \rightsquigarrow b_j^\ell$. We want to stay at the bottom layer if we are already there, so we also encode at $b_i^\ell \rightsquigarrow b_j^\ell$. On all other levels $k \neq \ell$, we simply stay at the level we are on, so we encode at the edges $t_i^k \rightsquigarrow t_j^k$, $c_i^k \rightsquigarrow c_j^k$, and $b_i^k \rightsquigarrow b_j^k$. Recall that we only added edges from the end of the *center* layer of each gadget to the super sink. We design our right bookend to include an encoding on each of these edges. The point of doing this is to enforce that zero-testing a monomial over encodings is only successful if each level is only included once among the encodings in the monomial. If level ℓ is included more than once, the iterated product of encodings will get stuck at the bottom layer of gadget ℓ and if the level is not included at all, the iterated product will still be at the top layer at the time of zero-testing. Both situations result in large random multiplications that ruin zero-test functionality. We present our construction below.



(a) Construction of the level graph to ensure all the levels are respected in a top-level encoding. We add a level gadget for each level, a “super” sink node \mathcal{T} , edge from the original sink to \mathcal{T} , and edges from all level gadgets to \mathcal{T} .



(b) Construction of Level Gadget 2, which ensures one and only one level-2 encoding is included in a top-level encoding. The dashed nodes and edges are never used in practice for this example construction.

Fig. 4. An example of how the level gadgets are used to ensure level-respecting encodings for a 5-node path graph with 6 levels and the given encodings.

$\gamma\text{-GGH15-AL2.KeyGen}(1^\lambda, G = (V, E), R = \mathbb{Z}, \kappa, \beta, k)$ ¹⁶:

Parameter Generation

- Sort the nodes of G in topological order v_1, \dots, v_d where v_1 is the unique source and v_d is the unique sink.
- Construct a new graph G' by augmenting G (which we now refer to as the “main branch”):
 1. Create a “super” sink node \mathcal{T} .
 2. For each $\ell \in [\kappa]$, construct “level gadget ℓ ” by creating a set of $3d$ nodes, denoted as $t_i^\ell, c_i^\ell, b_i^\ell$ for $i = 1, \dots, d$. For each edge $v_i \rightarrow v_j \in E$, create five edges $t_i^\ell \rightarrow t_j^\ell, t_i^\ell \rightarrow c_j^\ell, c_i^\ell \rightarrow c_j^\ell, c_i^\ell \rightarrow b_j^\ell, b_i^\ell \rightarrow b_j^\ell$.
 3. Create edges $c_d^\ell \rightarrow \mathcal{T}$ for all $\ell \in [\kappa]$, and edge $v_d \rightarrow \mathcal{T}$.
 4. See $\gamma\text{-GGH15-AL}$ for how we choose the parameters $n, w, n', m, q, \sigma, \chi, B$ and what they mean. We make three changes, namely we must now choose $m = \Theta((3\kappa + 1)n \log(q))$ for trapdoor functionality, take $\sigma = \omega(\text{poly}(\lambda_{\text{SZ}}))$ to ensure that polynomials over pre-encodings are zero with negligible probability, and multiply an extra $\kappa + 1$ factor into the zero-test bound B .

Instance Generation

- For $i \in [d - 1]$, sample $(\mathbf{A}_i, \tau_i) \leftarrow \text{TrapSam}(1^{n(3\kappa+1)}, 1^m, q)$
- Sample $(\mathbf{A}_d, \tau_d) \leftarrow \text{TrapSam}(1^{n(\kappa+1)}, 1^m, q)$
- Parse each \mathbf{A}_i for $i \in [d - 1]$ as a stack of $3\kappa + 1$ matrices of dimension $n \times m$ and \mathbf{A}_d as a stack of $\kappa + 1$ matrices of dimension $n \times m$:

$$\mathbf{A}_i := \begin{bmatrix} \mathbf{A}_{v_i} \\ \mathbf{A}_{t_i^1} \\ \mathbf{A}_{c_i^1} \\ \mathbf{A}_{b_i^1} \\ \vdots \\ \mathbf{A}_{t_i^\kappa} \\ \mathbf{A}_{c_i^\kappa} \\ \mathbf{A}_{b_i^\kappa} \end{bmatrix} \quad \mathbf{A}_d = \begin{bmatrix} \mathbf{A}_{v_d} \\ \mathbf{A}_{c_d^1} \\ \mathbf{A}_{c_d^2} \\ \vdots \\ \mathbf{A}_{c_d^\kappa} \end{bmatrix}$$

- We also create ‘dummy’ \mathbf{A} matrices at the final layer of the graph by drawing $\mathbf{A}_{t_d^\ell}, \mathbf{A}_{b_d^\ell}$ uniform from $\mathbb{Z}_q^{n \times m}$ for $\ell \in [\kappa]$. These fill out the d -th stack of $3\kappa + 1$ matrices and will be necessary for encoding on a path ending at node d . They will also be crucial for proving that the construction enforces level constraints. Essentially, zero-testing any non-level-respecting monomial encoded on a source-to-sink path in the main branch will necessarily involve multiplication with one of these dummy matrices, irrevocably altering the result.
- Sample a uniform $\mathbf{A}^* \leftarrow \mathbb{Z}_q^{n \times m}$ which we associate with the super sink node \mathcal{T}

Bookend Generation

- (Left bookend) We enforce that zero-testing happens with respect to the source node in the main branch as well as all the t_1^ℓ nodes from the level gadgets. An encoding can only be successfully zero-tested if it is an encoding of zero with respect to all of these nodes simultaneously. Thus we’ll define the “source” \mathbf{A} -matrix as follows

$$\mathbf{A}_S := \begin{bmatrix} \mathbf{A}_{v_1} \\ \mathbf{A}_{t_1^1} \\ \vdots \\ \mathbf{A}_{t_1^\kappa} \end{bmatrix}$$

¹⁶ κ is the number of asymmetric levels, β is a bound on size of plaintext entries, and k is the dimension of the block diagonal entries we will add in the encoding procedure.

- (Right bookend) Here we form an encoding from the center layer of each level gadget to the super sink.

Sample error matrix $\mathbf{E}_i^* \leftarrow (\chi)^{n \times m}$ for $i = 0, \dots, \kappa$, and let $\mathbf{S}^* = \begin{bmatrix} \mathbf{I}^{wn' \times wn'} \\ \mathbf{0}^{n' \times n'} \end{bmatrix}$.

$$\mathbf{D}^* \leftarrow \text{SampleD}(\mathbf{A}_d, \tau_d, \begin{bmatrix} \mathbf{S}^* \cdot \mathbf{A}^* + \mathbf{E}_0^* \\ \mathbf{S}^* \cdot \mathbf{A}^* + \mathbf{E}_1^* \\ \vdots \\ \mathbf{S}^* \cdot \mathbf{A}^* + \mathbf{E}_\kappa^* \end{bmatrix}, \sigma)$$

which is a simultaneous encoding of \mathbf{S}^* at all edges into the super sink (coming from node v_d and the set of nodes $\{c_d^\ell\}_{\ell \in [\kappa]}$). The lower block of zeros will serve to zero out the random \mathbf{P} matrices we add while encoding plaintexts.

- Sample uniform $\mathbf{v}' \in \mathbb{Z}_\nu^{n(\kappa+1)}$, $\mathbf{w}' \in \mathbb{Z}_\nu^m$, uniform $\mathbf{b}_v \in \mathbb{Z}_\nu^k$, $\mathbf{b}_w \in \mathbb{Z}_\nu^k$ and compute the bookends

$$\mathbf{v} = [\mathbf{v}' \cdot \mathbf{A}_1 | \mathbf{b}_v] \cdot \mathbf{R}_1, \quad \mathbf{w} = \mathbf{R}_d^{-1} \cdot \begin{bmatrix} \mathbf{D}^* \cdot \mathbf{w}'^\top \\ \mathbf{b}_w^\top \end{bmatrix}$$

Output

- Public parameters $\text{pp} = \{n, w, n', m, k, q, \sigma, \chi, \kappa, B, \mathbf{v}, \mathbf{w}\}$
- Secret parameters $\text{sp} = \{\mathbf{A}_i, \tau_i, \mathbf{R}_i\}_{i \in [d]}$

γ -GGH15-AL2.Enc(pp, $\mathbf{M} \in \mathbb{Z}_\beta^{w \times w}$, $i \rightsquigarrow j$, $L \subseteq [\kappa]$):

- Draw $\mathbf{P} \leftarrow D_{\mathbb{Z}, \sigma}^{n' \times n'}$, $\mathbf{E}_j \leftarrow (\chi)^{n \times m}$ for $j = 0, \dots, 3\kappa$, let

$$\mathbf{S} = \begin{bmatrix} \mathbf{M} \otimes \mathbf{P} \\ \mathbf{P} \end{bmatrix}$$

and $\bar{\mathbf{S}} = \mathbf{S} \cdot \mathbf{A}_{v_j} + \mathbf{E}_0$

- In addition to encoding \mathbf{S} on the main branch, it is also encoded on each of the level gadgets. For level gadget ℓ , if $\ell \in L$, \mathbf{M} is encoded on paths $t_i^\ell \rightarrow c_j^\ell$, $c_i^\ell \rightarrow b_j^\ell$, and $b_i^\ell \rightarrow b_j^\ell$. If $\ell \notin L$, \mathbf{M} is encoded on paths $t_i^\ell \rightarrow t_j^\ell$, $c_i^\ell \rightarrow c_j^\ell$ and $b_i^\ell \rightarrow b_j^\ell$. Thus, we'll define

$$\bar{\mathbf{S}}_\ell := \begin{cases} \begin{bmatrix} \mathbf{S} \cdot \mathbf{A}_{c_j^\ell} + \mathbf{E}_{3\ell-2} \\ \mathbf{S} \cdot \mathbf{A}_{c_j^\ell} + \mathbf{E}_{3\ell-1} \\ \mathbf{S} \cdot \mathbf{A}_{b_j^\ell} + \mathbf{E}_{3\ell} \end{bmatrix} & \ell \in L \\ \begin{bmatrix} \mathbf{S} \cdot \mathbf{A}_{t_j^\ell} + \mathbf{E}_{3\ell-2} \\ \mathbf{S} \cdot \mathbf{A}_{c_j^\ell} + \mathbf{E}_{3\ell-1} \\ \mathbf{S} \cdot \mathbf{A}_{b_j^\ell} + \mathbf{E}_{3\ell} \end{bmatrix} & \ell \notin L \end{cases}$$

- Compute $\mathbf{D} \leftarrow \text{SampleD}(\mathbf{A}_i, \tau_i, \begin{bmatrix} \bar{\mathbf{S}} \\ \bar{\mathbf{S}}_1 \\ \vdots \\ \bar{\mathbf{S}}_\kappa \end{bmatrix}, \sigma)$
- Draw uniform $\mathbf{B} \leftarrow \mathbb{Z}_\nu^{k \times k}$ and output the encoding

$$\mathbf{C} = \mathbf{R}_i^{-1} \cdot \begin{bmatrix} \mathbf{D} \\ \mathbf{B} \end{bmatrix} \cdot \mathbf{R}_j$$

γ -GGH15-AL2.ZeroTest(pp, \mathbf{C}):

- Return true if $|\mathbf{v} \cdot \mathbf{C} \cdot \mathbf{w}^\top| \leq B$. If \mathbf{C} is a level-respecting source-to-sink encoding of zero on the main branch, then it is actually a simultaneous encoding of zero on the paths $v_1 \rightsquigarrow v_d, t_1^1 \rightsquigarrow c_d^1, \dots, t_1^\kappa \rightsquigarrow c_d^\kappa$. The structure of \mathbf{v} and \mathbf{w} then enforces that the result of this multiplication is the sum of $\kappa+1$ successful zero-tests over regular GGH15. Correctness follows from analysis in γ -GGH15-AL and the extra $\kappa+1$ term in the zero-test bound.

Lemma 13. *Let \mathcal{A} be a successful adversary in the GGH15 Zeroizing Model instantiated with γ -GGH15-AL. Then there exists a successful adversary \mathcal{A}' in the Level-Restricted GGH15 Zeroizing Model instantiated with γ -GGH15-AL.*

Proof. Let \mathcal{A} be an adversary in the GGH15 Zeroizing Model and say that it submits a polynomial $p(\{\mathbf{C}_i\}_i)$ for zero-testing that contains some non-level-respecting monomials. The intuition behind this proof is that any non-level-respecting monomial will not be encoded relative to the node c_d^ℓ for some ℓ and thus multiplying this part by \mathbf{D}^* during zero-test will not zero out the bottom n' rows of the pre-encoding as expected. In fact, this multiplication will include elements of one of the large and random ‘dummy’ \mathbf{A} matrices and will thus be close to uniformly random, making a successful zero-test possible only with negligible probability.

There are two cases, but both lead to essentially the same proof. For the first case, assume that there exists some level $\ell \in [\kappa]$ such that there exists monomials in p that do not contain any encoding that has ℓ as a part of its level subset. Our goal is to show that p will only give a successful zero-test with negligible probability and we will show this by eventually considering p as a polynomial over the formal variables in the matrix $\mathbf{A}_{t_d^\ell}$. Denote by \hat{p} the restriction of p to monomials which don’t include any encodings at level ℓ . We expand out the polynomial that results from zero-testing p and push terms that are independent of $A_{t_d^\ell}$ into a constant term \mathcal{K} , obtaining

$$\mathbf{v}' \hat{p}(\{\mathbf{S}_i\}_i) \mathbf{A}_{t_d^\ell} \mathbf{D}^* \mathbf{w}'^\top + \mathcal{K}$$

where \mathcal{K} includes terms of p not in \hat{p} , block diagonal matrices \mathbf{B} , error matrices \mathbf{E} and encodings on level gadgets other than ℓ .

Now consider this polynomial over formal variables $\{\hat{a}_i\}$ substituted for each element of $\mathbf{A}_{t_d^\ell}$. First, notice that all the coefficients and the constant term are derived from quantities that are independent of $\mathbf{A}_{t_d^\ell}$. By the Schwartz-Zippel lemma and the union bound, for p to be a successful zero-test, this polynomial must be identically zero over the $\{\hat{a}_i\}$ variables. However, notice that every element of \mathbf{v}' and $\mathbf{D}^* \cdot \mathbf{w}'^\top$ (determined at KeyGen time) is non-zero with overwhelming probability. So, the only way this polynomial is identically zero is if $\hat{p}(\{\mathbf{S}_i\}_i)$ is the all-zeros matrix. But since the \mathbf{S}_i matrices include the random \mathbf{P}_i matrices as lower block diagonals, whose entries are drawn independently from a super-poly in λ size set, we can apply the Schwartz-Zippel lemma again to conclude that the set of monomials we are considering (that don’t include level ℓ) are actually identically zero over the encodings. The same proof applies for every level and for the second case where we consider a level read multiple times. \square