Attacks on Multilinear Maps

by

William Cyr

Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degree of Masters of Engineering in Computer Science and Engineering at the MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2015

© William Cyr, MMXV. All rights reserved.

The author hereby grants to MIT permission to reproduce and to distribute publicly paper and electronic copies of this thesis document in whole or in part in any medium now known or hereafter created.

Author

Department of Electrical Engineering and Computer Science

May 19, 2015

Certified by

Vinod Vaikuntanathan
Associate Professor
Thesis Supervisor

Accepted by

Albert Meyer
Chairman, Masters of Engineering Thesis Committee
Attacks on Multilinear Maps

by

William Cyr

Submitted to the Department of Electrical Engineering and Computer Science
on May 19, 2015, in partial fulfillment of the
requirements for the degree of
Masters of Engineering in Computer Science and Engineering

Abstract

In this thesis, I explore the current multilinear map candidates and attacks against them. This involves analyzing the three proposals for multilinear maps which are based on ideal lattices, integers, and standard lattices. The attacks exploit the geometry of the lattices and linearity of the integers to break security. I also compare the applications of these schemes with what is required for attacks. Key agreement seems to need certain features of multilinear maps which expose vulnerabilities while other applications like indistinguishability obfuscation. I analyze the attacks against these maps and show why they are not able to break the program obfuscation application of multilinear maps.

Thesis Supervisor: Vinod Vaikuntanathan
Title: Associate Professor
## Contents

1 Introduction ........................................... 7
   1.1 Lattices and Ideal Lattices ..................... 7
   1.2 Multilinear Maps ................................ 9
   1.3 Graded Encoding Scheme ......................... 9

2 Candidate Multilinear Maps .......................... 13
   2.1 Multilinear Map with Ideal Lattices ............. 13
      2.1.1 Eliminating the Need For Zero .............. 15
   2.2 Multilinear Map over the Integers ............... 16
      2.2.1 Nonlinear Zero Testing ...................... 18
   2.3 Multilinear Map over Standard Lattices .......... 18

3 Applications ........................................ 21
   3.1 Key Agreement .................................... 21
      3.1.1 GGH13 ....................................... 21
      3.1.2 GGH15 ....................................... 22
   3.2 Program Obfuscation ............................... 24
      3.2.1 Creating a Branching Program ............... 25
      3.2.2 Obfuscating a Branching Program ............ 25

4 Attacks .............................................. 29
   4.1 GGH13 ........................................... 29
4.1.1 Attacks given encodings of 0 ........................................ 29
4.1.2 Breaking GGH13 with a small multiple of $1/h$ ............... 33
4.2 CLT14 ................................................................. 34
4.3 GGH15 ................................................................. 35
4.4 Other Attacks .......................................................... 36
   4.4.1 SVP .............................................................. 36
   4.4.2 Bernstein ......................................................... 36
   4.4.3 Soliloquy ........................................................ 37
4.5 Short Generator From Principal Ideal .................................. 37
4.6 Current Status ......................................................... 39
   4.6.1 Requirements For Attack ...................................... 39
   4.6.2 Requirements For Construction ............................... 39
4.7 Open Problems ....................................................... 39
4.8 Conclusion ............................................................ 40
Chapter 1

Introduction

Multilinear maps and graded encoding schemes are very desirable objects in cryptography. From them, we can create program obfuscators, attribute based encryption, multiparty key exchange, and many more other cryptographic schemes. However, there have been only a few proposals for multilinear maps. Recently [GGH12] proposed a candidate construction based on ideal lattices. Another proposed candidate is [CLT13] which works over the integers. This however was broken but a fix has been proposed [CLT15] which makes the scheme resistant to the known attacks. Finally the third scheme that has been proposed is [GGH14] which uses standard lattices and relies on the hardness of a different problem, one closely related to learning with errors. These candidates may allow us to realize indistinguishability obfuscation, a notion from which many other cryptographic tools can be built. Because of this, it is important to study the security of these proposals and their variants and to test their resilience to attacks.

1.1 Lattices and Ideal Lattices

A lattice $L \subset \mathbb{R}^n$ is an additive subgroup of $\mathbb{R}^n$. It is defined by a basis $L = \{\sum_{i=1}^{n} z_i b_i : z_i \in \mathbb{Z}\}$ where the points $b_i$ are linearly independent points. We can write the vectors $b_i$ as columns in the matrix $B \in \mathbb{R}^{n \times n}$ so the lattice is $L = \{Bz : z \in \mathbb{Z}^n\}$. There are many
equivalent bases for a lattice and in general, we are not to assume that the basis is reduced. In fact, a lattice problem conjectured to be hard is finding a short basis for a given lattice. Two other common hard problems associated with a lattice are the Shortest Vector Problem (SVP) and the Closest Vector Problem (CVP). In SVP, we are given a basis for a lattice and must find the shortest non-zero vector in the lattice. In CVP, we are given a basis and a target point and must find the lattice vector that is closest to the point. The best known algorithms for these run in exponential time for standard lattices, so it is common to try to reduce security to one of these problems.

Lattices can be described by their matrix $B \in \mathbb{R}^{n \times n}$ but sometimes that is too big, or does not have all the desired properties. A standard lattice is an additive group. However, sometimes we want multiplication, and that is why ideal lattices are useful. In most applications, we will consider $n$ to be a power of two and use the cyclotomic ring $R = \mathbb{Z}[X]/(X^n+1)$ where an element in the ring, $u \in R$, is represented by its coefficient vector. Then $R$ is a lattice with addition and multiplication defined as polynomial addition and multiplication in the ring. This is called an ideal lattice and so is any sublattice that is closed under multiplication and addition.

Sometimes we will be concerned with a principal ideal in the lattice. This is an ideal with a generator $g \in R$ such that $(g)$ is the sublattice we want. This can be viewed as both an ideal and a lattice since $g, gX, gX^2, \ldots gX^{n-1}$ are also in the lattice as well as any linear combination of those. There is not always a single generator and in cases where there is one, it is not necessarily given, so this is a very special case although it does come up frequently. These lattices have another computational problem. The Principal Ideal Problem is the task of finding a generator for an ideal given a basis for the ideal. This is usually a problem for ideals, but since an ideal lattice has the properties of an ideal, it also applies. The ability to do multiplication in an ideal lattice allows more efficient operations in the scheme, but also gives the adversary additional power.
1.2 Multilinear Maps

A multilinear map is a mapping that is linear in all of its inputs. It takes inputs from \( k \) different groups \( G_1, \ldots, G_k \) of order \( p \), and produces an output in another group \( G_T \). It is linear in all of its inputs so that \( e(g_1, \ldots, \alpha g_i, \ldots g_k) = \alpha e(g_1, \ldots, g_i, \ldots g_k) \) for all \( i \). It is also non-degenerate.

It needs to support a few operations. There needs to be a randomized instance generation to construct the multilinear map. This takes the degree of multilinearity, \( 1^k \), and the security parameter, \( 1^\lambda \) as input and generates the instance of the multilinear map. It needs to support encoding so a bit string can be mapped into a group element. It needs to support group operations in each of the groups involved. And it needs to have a multilinear evaluation function, \( e \) which maps \( G_1 \times \cdots \times G_k \rightarrow G_T \).

These are a useful construction in cryptography since many applications can come from them. Therefore, we need to consider the hardness associated with them. There is the discrete log problem. Another problem is Decision-Diffie-Helman (DDH). An adversary should not be able to distinguish between the correct output and a random output.

1.3 Graded Encoding Scheme

We will not always work directly with multilinear maps, but rather a different construction called a Graded Encoding Scheme. They are similar since they are linear in up to \( k \) inputs, but a graded encoding scheme allows for many different encodings of the same element at a given level while a multilinear map requires just one. A graded encoding scheme encodes elements in such a way that encodings have an associated level. We denote the set of all encodings of \( \alpha \in \mathbb{R} \) at level \( i \leq k \) as \( S^\alpha_i \). Encodings at the same level can be added to produce an encoding of the sum at the same level and encodings can be multiplied to produce an encoding of the product at the sum of their levels.

All these actions operations occur on encodings, not plaintext. In fact, there is no guarantee that we can ever decode any of the encodings. There is however a zero testing
parameter and an extraction parameter. At a certain fixed level $k$ determined in the instance generation, these parameters act as a defective decryption key. They can return either whether an encoding is zero in the zero testing case or a canonical representation of the element that is encoded in the extraction case. They are like Fully Homomorphic Encryption except the decryption key is public and less powerful and as a result, we cannot decode, but anyone can extract certain information. There are 7 functions that a Graded Encoding Scheme provides.

**Instance Generation** Takes as input $1^\lambda$ which is the security parameter and $1^\kappa$ which is the level for zero testing. This outputs the params needed and the zero tester $p_{zt}$

**Ring Sampler** This produces a level 0 encoding of an element $\alpha$ where $\alpha$ is an element sampled uniformly from the ring.

**Encode** Takes an $\alpha$ at level 0 and a desired level $i$ and produces an encoding of $\alpha$ at level $i$.

**Add** Takes two encodings at the same level, $u \in S_i^{\alpha_1}$, $v \in S_i^{\alpha_2}$ and outputs an encoding of their sum at the same level, $u + v \in S_i^{\alpha_1 + \alpha_2}$.

**Multiply** Takes two encodings $u \in S_i^{\alpha_1}$, $v \in S_i^{\alpha_2}$ and outputs an encoding of their product at the sum of their levels, $u \times v \in S_i^{\alpha_1 + \alpha_2}$.

**Zero-test** Takes an encoding $u$ and decides if it is an encoding of zero at level $k$, $u \in S_k^0$.

**Extract** Takes an encoding $u$ at level $k$ and produces a canonical representation of $u$ such that if $u, v \in S_k^0$, then $\text{extract}(u) = \text{extract}(v)$. And the range of the extract function is nearly uniform over $0, 1^\lambda$.

These are the function in the ideal model, but in practice, we allow the zero-test to have false positives with low probability and the extract does not necessarily guarantee a different output for encodings of different elements.
The hard problems for graded encodings are similar to those for multilinear maps. An adversary should not be able to take the discrete log of an element. So they should not be able to reduce the level of any element. An adversary should not be able to break DDH. Given $k + 1$ level 1 encodings of $\alpha_1, \ldots, \alpha_{k+1}$, an adversary should not be able to distinguish a level $k$ encoding of $\prod_{i=1}^{k+1} \alpha_i$ and a random level $k$ encoding with non-negligible advantage. Similarly it would be nice for CDH to also hold so that an adversary could not compute an equivalent encoding. These assumptions of our Graded Encoding schemes will be used to provide security in the key exchange and program obfuscation applications.
Chapter 2

Candidate Multilinear Maps

There are currently three candidates for multilinear maps. The first is based on ideal lattices, the second uses the integers, and the third works with standard lattices and directed acyclic graphs. In the description of these constructions, the details on how to set parameters such as how large the primes should be and what constitutes a small or somewhat small norm as well as some proofs of correctness are omitted from this paper. Those details are in the papers which propose these schemes.

2.1 Multilinear Map with Ideal Lattices

The first candidate construction is based on ideal lattices and the hardness assumption is somewhat related to the shortest vector problem. Though there is no known reduction to it, attacks against SVP and variants of SVP seem to reveal certain secret parameters involved in this construction.

This construction works over a cyclotomic ring $R = \mathbb{Z}[X]/(X^n + 1)$ where the degree is determined by the security parameter $\lambda$. The ring is then taken modulo a large prime $q$ so that $R_q = R/qR$. Most operations will occur over this ring. A secret generator $g$ of a smaller ring will be used and encodings will be cosets of this small hidden ideal. In addition, an invertible element $z \in R$ will be used so that every level $\ell$ encoding has a factor of $z^{-\ell}$ which
enforces the level restrictions for addition, multiplication, and zero testing. An encoding of \(\alpha\) at level \(\ell\) will be an element of the form \(\frac{rg + \alpha}{z^\ell}\). Zero testing will occur at level \(k\) and it will consist of multiplying by the zero testing parameter and then checking whether the result is small or not.

**Instance Generation** Given \(1^\lambda\), the security parameter and \(1^k\), the degree of multilinearity, set \(n\) to be a power of two. For details on how to set the parameters see [GGH12]. This defines \(R = \mathbb{Z}[X]/(X^n + 1)\) and \(R_q = R/qR\). Choose \(z\) uniformly at random from \(R_q\) and choose \(g\) from \(R_q\) such that \(g\) and \(g^{-1} \in \mathbb{Q}[X]/(X^n + 1)\) are short. Publish random small multiples of \(g/z\). These are the encodings of zero at level one used for randomizing. Also, publish a randomized level 1 encoding of 1, call it \(y\). Construct the zero testing parameter by sampling a random \(h\) from \(R_q\) such that \(|h| \approx \sqrt{q}\). Publish \(p_{zt} = \frac{h + g}{g}\) as the zero testing parameter.

**Sampling** To sample level zero encodings, we can sample a random small element from \(R_q\). It is a short representative of its coset.

**Encode** To encode an element, \(u\), from level 0 to level \(i\), we multiply it by \(y^i\). Since multiplying results in an encoding of the product, multiplying by \(y\) does not change the value encoded and because it adds the levels, the level is increased. However, this is easy to reverse, so we may need to randomize the encoding. This is achieved by adding a random multiple of an encoding of zero which is given in the params.

**Addition** To add encodings of \(u \in S_i^{\alpha_1}\) and \(v \in S_i^{\alpha_2}\), add the ring elements. Because \(u = \frac{\alpha_1 + r_1 s g}{z^i}\) and \(v = \frac{\alpha_2 + r_2 s g}{z^i}\), their sum is an encoding of \(\alpha_1 + \alpha_2\) at level \(i\). Because the noise in each encoding is small, the noise in the sum is also small, so it is a valid encoding.

**Multiplication** To multiply encodings of \(u \in S_i^{\alpha_1}\) and \(v \in S_i^{\alpha_2}\) multiply the ring elements. Because \(u = \frac{\alpha_1 + r_1 s g}{z^i}\) and \(v = \frac{\alpha_2 + r_2 s g}{z^i}\), their product is an encoding of \(\alpha_1 \times \alpha_2\) at level \(i\). Because the noise in each encoding is small, the noise in the sum is also small, so it is a valid encoding.
Zero Testing To zero test an encoding, \( u \), multiply by the zero testing parameter \( p_{zt} \) if the result is small then output 0. Otherwise output 1. Intuitively, if it is an encoding of zero, it has a factor of \( \frac{q}{zt} \) which will cancel with the zero testing parameter leaving only small factors and \( h \) which is "somewhat small" enough. However if it is not an encoding of 0, then the factor of \( \frac{1}{q} \) in the zero testing parameter will not cancel and the result is large with high probability.

Extract To extract bits from an encoding, multiply by the zero testing parameter and return the high bits. For different encodings of the same element, this will give the same high order bits.

The hardness of this scheme seems related to hard problems in ideal lattices, although there is no reduction to those problems. For example, an adversary that can solve the SVP can break GGH13, but solving GGH13 does not imply being able to solve SVP.

2.1.1 Eliminating the Need For Zero

When the attacks on CLT14 came out, the question of whether encodings of zero are really necessary came about. This led to the proposal of a modification of GGH13 in [GHMS14]. The idea is that we can use randomizing matrices to enforce the order in which the encodings are multiplied to try to prevent intermediate encodings of zero from arising.

Working over the same ring \( R \), an encoding of \( \alpha \) will be a matrix \( A \) such that \( \alpha \) is an eigenvalue of \( A \) for an eigenvector \( s \) which is public. Instead of publishing \( A \) directly, we pick a random matrix \( T \) for each level \( i \) and to make \( A \) and encoding of \( \alpha \) at level \( i \), use \( T_{i-1} \times A \times T_i^{-1} \). This enforces the order in which encodings are multiplied since the randomizing matrices will cancel when multiplied in the correct order. The zero testing parameter has to be augmented to cancel \( T_0 \) and \( T_k^{-1} \), but that is easy to do. It is not obvious how to get a native GGH13 encoding of zero given a matrix that encodes zero because the randomizing matrices enforce the order in which the encodings are multiplied. They are random, so this tries to make the encodings useless on their own without cancelling.
the $T_i$. So we can publish randomizing matrices without publishing native encodings of zero. However, it turns out that the same attacks that work against regular GGH13 that use low level encodings of zero also apply to this modified scheme. A similar idea will be used in program obfuscation, however in that situation, encodings of zero need not be public, so the attacks do not work.

2.2 Multilinear Map over the Integers

The next proposal for a multilinear map uses the integers for its encodings [CLT13]. It is very similar to the original GGH13 scheme, but it has some additional vulnerabilities that can be exploited because of properties of the integers. The original proposal was broken, then there were fixes proposed also which have since been broken. Most recently, the scheme has been changed to have a nonlinear zero testing parameter which fixes the problems that allowed the prior attacks.

The scheme encodes vectors of integers. It uses large primes as moduli for each slot in the vector and the encodings are of cosets of smaller primes. The large primes correspond to $q$, the large modulus in GGH and the small primes correspond to the secret element $g$ in GGH13. Similarly, an encoding at level $k$ has a factor of $\frac{1}{z^k}$ for a random $z$ that is cancelled by the zero testing parameter. An encoding of $\alpha$ at level $j$ will be an integer such that it is $\frac{r_ig_i+\alpha}{z^j} \mod p_i$ for each $i$.

Instance Generation Generate $n$ random large primes $p_i$ and publish their product $x_0 = \prod_{i=1}^{n} p_i$. Generate a random invertible integer $z$ modulo $x_0$. Generate $n$ small random primes $g_i$. Generate an encoding of 1 and publish it as $y$. Generate many encodings of 0 at level 1 and publish them as $x_j$. Publish many random encodings of random elements at level 0 and publish them as $x'_j$. Also, generate and publish the zero testing parameter which is described in the zero testing section.

Sampling To sample a random level 0 encoding, use the $x'_j$ which are encodings of random elements at level 0. Take a random subset sum of these to generate an encoding of
a random element. This produces a distribution of encodings which can be made statistically close to uniform.

**Encode** Similarly to GGH13, to increase the level of an encoding, we multiply by the public encoding of 1 at level 1. This increases the level of the encoding and preserves the element that is encoded. To see this, notice that
\[ y \equiv r_i g_i + 1 \mod p_i \]
for all \( 1 \leq i \leq n \) where \( r_i \) are small random integers. Multiplying by this preserves the element encoded and multiplies by \( \frac{1}{z} \) to increase the level. This may need to be re-randomized for some applications so an adversary cannot simply divide by \( y \) to undo the encoding.

**Addition** Addition is addition over the integers. Suppose we are adding \( \frac{r_i g_i + \alpha_i}{z^k} \mod p_i \) and \( \frac{r'_i g_i + \beta_i}{z^{k'}} \mod p_i \). Then the sum will be \( \frac{(r_i + r'_i) g_i + (\alpha_i + \beta_i)}{z^{k + k'}} \mod p_i \) which is an encoding of their sum at the same level.

**Multiplication** Multiplication is multiplication over the integers. Suppose we are multiplying \( \frac{r_i g_i + \alpha_i}{z^k} \mod p_i \) and \( \frac{r'_i g_i + \beta_i}{z^{k'}} \mod p_i \). Then the product will be \( \frac{r''_i g_i + (\alpha_i + \beta_i)}{z^{k + k'}} \mod p_i \) which is an encoding of their product at the level which is the sum of their levels.

**Zero Test** To generate the zero testing parameter, randomly sample a somewhat small matrix of integers that is invertible over the integers, call it \( H \). The zero testing parameter is defined as
\[
(p_{zt})_j = \sum_{i=1}^{n} h_{ij} (z^{k_i} g_i^{-1} \cdot \prod_{i' \neq i} p_{i'}) \mod x_0
\]
If \( c \times p_{zt} \mod x_0 \) is small, then it is zero, otherwise, \( c \) is an encoding of non-zero. This is similar to GGH13 since there is a somewhat small element multiplied by \( \frac{z^{k_i} g_i^{-1}}{g} \) which will cancel only if the level is correct and the encoding has a factor of \( g \). If it doesn’t cancel, then the result is not small.

**Extract** Similar to GGH13, the extraction consists of multiplying by the zero testing parameter and extracting the high order bits from the result. This can be done in each
2.2.1 Nonlinear Zero Testing

There are attacks against CLT which break it by using the zero testing parameter to reveal more than it is supposed to. As a result, a fix was proposed [CLT15] which makes the zero testing process non-linear, so an adversary cannot just get lots of relations and solve linear equations. The new zero testing procedure uses another even larger modulus and argues that the original scheme is just a weak particular case of the new one. The difference is some parameter $a$ which is non-linear in the rest of the variables and in the case of CLT14 was just zero, so it did not add complexity.

Additionally, the public modulus is now made private since attacks were able to decompose it into the product of its primes and use that to break the scheme. Instead of being able to reduce modulo $x_0$, there is a process for reducing via a ladder of encodings that reduce back down to the size of $x_0$ while preserving the cosets that are encoded. It maintains the size of encodings by reducing them one bit at a time by publishing random multiples of $x_0$ of varying sizes instead of publishing $x_0$. This also keeps the errors small while preserving the coset.

This attempts to fix the linearity that was used to break CLT13 with the Cheon et al attack [CHL+14]. They managed to recover all secret parameters because they could use the zero testing parameter to give more information than it was supposed to. The changes make the zero testing parameter reveal less information, so they believe this makes it harder to break. There currently is not a way to break this fixed scheme.

2.3 Multilinear Map over Standard Lattices

The most recent candidate multilinear map is GGH15. It is closer to standard lattice assumptions than the other two. The difference is that a level is relative to a path in a directed acyclic graph and we can add encodings if they are on the same path and we can
multiply them if the source of one is the destination for the other, so the paths connect. The
hardness assumption is not quite the same as learning with errors, but it is a very close vari-
ant. The scheme uses approximate eigenvectors, but increases dimension to use approximate
eigenspaces so that a zero testing parameter can be provided without breaking the scheme.
This does not allow rerandomization of encodings like the other schemes, but that does not
seem necessary. Also, there is no need to publish encodings of 1 and 0 at low levels in this
scheme which may make it more secure than the other schemes since those are crucial in
the attacks against those schemes. Another difference is that it is not possible to publicly
encode elements. A set of public random encodings relative to each edge is given and the
user generates a random subset sum of these encodings to generate a new random encoding.

**Instance Generation** Generate the underlying DAG. Setup all the parameters for the
scheme. Using trapdoor sampling, create a matrix $A_v$ for every vertex sampled so that
there is a trapdoor [GPV07] that will be used when creating the published encodings.

**Sampling** Sample a small matrix $S$ which will be an LWE secret and this is the plaintext.

**Encode** Encoding is done in the setup phase to create encodings of random elements. These
are published and public encoding is done via a random subset sum of these public
encodings. However, first we must generate the random encodings relative to each edge
$(u, v)$. Given matrices $A_u$ and $A_v$ and a matrix to encode $S$. Sample a small error
vector $E$ and using the trapdoor, solve for the encoding, $D$, such that $DA_u = A_vS+E$.

**Addition** Addition of encodings is addition of matrices. This only works if the encodings
are have paths with the same start and end. Elements encoding $S_1$ and $S_2$ satisfy
$D_1A_u = A_vS_1 + E_1$ and $D_2A_u = A_vS_2 + E_2$. So adding the matrices will satisfy
$(D_1 + D_2)A_u = A_v(S_1 + S_2) + (E_1 + E_2)$ which is an encoding of the sum with respect
to the same path.

**Multiplication** Multiplication of encodings is multiplication of matrices. This only works
if the encodings are with respect to connecting paths. Elements encoding $S_1$ and $S_2$
satisfy $D_1A_u = A_vS_1 + E_1$ and $D_2A_v = A_wS_2 + E_2$. So multiplying the matrices will satisfy $(D_2D_1)A_u = A_w(S_1S_2) + (E_2S_1) + (D_2E_1)$ which is an encoding of the product with respect to the same path.

**Zero Test** We know that $DA_u = A_vS + E$. If $S$ is an encoding of zero, then $DA_u = E$ which is small, so it is an encoding of zero if $DA_u$ is small. Because $A_v$ is a random matrix, the product will be large if $S$ is not an encoding of zero.

**Extract** Extract takes the most significant bits of $DA_u$. Similarly to the other schemes, the extraction is the most significant bits of zero testing for non-zero elements.

This scheme nicely is able to hide encodings of 0 and 1 which present weaknesses in other schemes. It also seems to have less structure for the adversary to use in attacks which makes it seem safer than the other candidates. There are still attacks on it similarly to the other candidates and they rely on the adversary getting intermediate encoding of 0 just as in the other constructions.
Chapter 3

Applications

Multilinear maps can be used to create other cryptographic tools. The first is key agreement. While it has long been known how to do two party key agreement, multilinear maps make it possible to do it for \( n \) parties. Program obfuscation is another cryptographic tool that can be realized from multilinear maps. Multilinear maps can be used to create an indistinguishability obfuscator [BGK+13] which in turn can be used to create many other cryptographic primitives. So the applications of multilinear maps are definitely important.

It is important in these applications to note which of the public parameters are required. These are the reason for which some of the parameters like the low level encodings of zero are provided in the schemes as described in their papers. Obfuscation seems to need to publish less public parameters than key agreement.

3.1 Key Agreement

3.1.1 GGH13

Suppose \( n \) people want to share a secret key using GGH13. The high level idea is that each participant samples a level zero encoding and raises the level to level 1. When everyone publishes their level 1 encoding, there are \( n \) of them, but each participant knows their own at level 0, so each participant can construct an encoding of the product at level \( n - 1 \). They
compute the same product, so they all extract the same bits.

**Setup** The scheme is set up so that if $n$ people want to share a secret, then the multilinear map has zero testing and extraction available at level $n - 1$.

**Publish** Each participant secretly samples a random level 0 encoding, $\alpha \in S_0^\alpha$. Then they multiply $\alpha$ by $y$, the encoding of 1 at level 1 to make an encoding of $\alpha$ at level 1. This is not enough however because an adversary could just divide by $y$ to compute the discrete log, which would break security. To prevent this, each participant adds a random subset sum of the public encodings of 0 at level 1. By doing this, an adversary can no longer recover a short representation of $\alpha$ simply by dividing by $y$. This is the rerandomization process which keeps the encoding equivalent because adding zero does not change the coset, but makes it no longer possible to divide by $y$. Each participant publishes their resulting level 1 encoding of their secret, $\beta = \alpha \times y + \sum r_i z_i \in S_1^\alpha$.

**KeyGen** To compute the shared key, each participant has their own $\alpha$ and level 1 encodings of every other participant’s $\beta$. Take the product of these to compute a level $n - 1$ encoding of the product of everyone’s secrets. Each participant uses extract on this product which can be done because the scheme was designed with $k = n - 1$. By the correctness of extraction, everyone computes the same shared key.

Security is similar to, but not exactly CDH on the multilinear map so that an adversary cannot compute an equivalent encoding of the product of $\alpha_i$. However, this is slightly different since the adversary only needs to compute the high bits of the product of an equivalent encoding and the zero testing parameter.

### 3.1.2 GGH15

In GGH15, we must first define the graph in the setup phase. To share a secret among $n$ people, make a single sink and $n$ distinct paths each with $n$ edges pointed towards the sink. Each participant will be assigned one of these paths. To generate the share to broadcast, a participant will randomly sample a secret, $\alpha$. Sample the same secret along the first edge of
Figure 3-1: Graph for 4 party key agreement

their path, the second along the next, and so on until having sampled the secret along each path. Broadcast the encoding of the secret that was sampled on everyone else’s branches.

To compute the shared key, note that the product of the encodings along each of the \( n \) paths is equal to the product of the sampled secrets. Because each participant has their own secret along the first edge of their path and all the rest were broadcasted, they can compute the product which follows the path all the way from source to sink. Since everyone’s product is the same and the sink is the same, the correctness of extract guarantees that everyone will compute the same shared key.

For example, Alice, Bob, Carol, and Dave want to compute a shared key together,

**Setup** Create a GGH15 instance using 3-1 as the graph.

**Publish**

1. Alice samples a random encoding along using a random subset sum of the public encodings along (1,2) and the same subset sum along the edges, (6,7) and (11,12) and (16,SINK). Alice publishes the encodings for (6,7) and (11,12) and (16,SINK).

2. Bob does the same sampling procedure along the edges, (2,3) and (7,8) and (12,SINK)
and (13,14). and publishes all but the encoding along (13,14).

3. Carol does the same sampling procedure along the edges, (3,4) and (8,SINK) and (9,10) and (14,15) and publishes all but the encoding along (9,10).

4. Dave does the same sampling procedure along the edges, (4,SINK) and (5,6) and (10,11) and (15,16) and publishes all but the encoding along (5,6).

**KeyGen**

1. Alice computes the encoding along the path from 1 to SINK and runs extract.

2. Bob computes the encoding along the path from 13 to SINK and runs extract.

3. Carol computes the encoding along the path from 9 to SINK and runs extract.

4. Dave computes the encoding along the path from 5 to SINK and runs extract.

Note that in this construction there was no need for publishing randomizing parameters. Because sampling is a subset sum of public encodings, an adversary cannot invert the encodings so rerandomizing is not necessary like in the other schemes. Participants in this scheme sample random encodings without knowing what element they are encoding. In other schemes, publishing the randomizing parameters consists of a level level encoding of zero which seems to give the adversary too much power.

### 3.2 Program Obfuscation

The Brakerski-Rothblum construction [BR13] allows us to achieve program obfuscation with these multilinear map candidates. There are two parts to the construction. First, the circuit is converted into a branching program which also computes the program. Then it uses the multilinear map to replace the permutation matrices which will obfuscate the program and guarantee the necessary security.
3.2.1 Creating a Branching Program

The first step is to convert the program into a branching program. This can be done with Barrington’s Theorem which says we can convert any circuit into a branching program as well as some bounds on the size of the program. The branching program consists of pairs of permutation matrices $M_{i,j}$ as in Figure 3-2. There is also a function $\ell$ which determines which matrices correspond to each input bit. So if $\ell(i) = k$, then the $i$th pair of matrices is controlled by the $k$th input bit and when evaluating, we will pick $M_{i,x_k}$ out of the pair where $x_k$ is the $k$th input bit. Evaluation consists of multiplying all the matrices corresponding to the input, one from each pair, and if the result is the identity, then output zero, and if the result is some other matrix, output 1. By construction, evaluation will either be the identity or some other fixed matrix.

There are other complications necessary like consistency checks which guarantee that when evaluating, if a bit corresponds to more than one pair of matrices, it must be the same in each case. This enforces that an adversary must use only valid inputs to the program. Once it is possible to obfuscate circuits in $\mathcal{NC}^1$, using homomorphic encryption, it is possible extend the result to obfuscate $P$.

3.2.2 Obfuscating a Branching Program

The branching program computes the circuit, but it is not yet obfuscated. To do this with GGH13, replace elements of the matrices with level one encodings of those elements. When
evaluating, we need to be able to add and multiply encodings which is possible, but the level at the end grows to $n$. We also publish a level $n$ encoding of the identity matrix. To check if the result of the computation is zero, subtract the public encoding of the identity matrix and zero test.

The construction goes one step further. It uses the Killian’s Randomization Technique [Kil88] to randomize the program matrices. To do this, sample $n+1$ random invertible matrices $R_i$. Instead of publishing $M_{i,j}$, we publish $R^{-1}_{i-1}M_{i,j}R_i$. Because we draw the matrices from uniform random, the resulting permutation matrices are each individually indistinguishable from random. When evaluating we need instead of an encoding of the identity matrix, an encoding for $R_0^{-1}IR_n$. Notice that this enforces a graph similar to GGH15. In order to cancel the randomizers, we need to multiply the permutation matrices in order, so we lose the flexibility of levels and are equivalent to the line graph for GGH15.

By using the randomizers, even with GGH13, there becomes an induced graph for levels. If we would use GGH15 as the multilinear map instead of GGH13, the DAG that it would use is also the line graph. The randomizers restrict the order and ways in which matrices can be multiplied because they do not commute and only cancel in one way. This reduces the ability of the adversary to try to get an encoding of zero.

Note that in this construction, the adversary does not need to be given the ability to encode. Also, the adversary is not given low level encodings of zero. All that is public are the randomized matrices corresponding to the obfuscated program, the zero testing parameter, and an encoding of the identity at the highest level. The public matrices are made indistinguishable from uniform random by the randomization process, so they do not seem to give the adversary much ability without running the obfuscated program. The adversary can compute encodings of zero at the highest level by running the obfuscated program for different inputs and subtracting the results. Since there are only two possible products of the encodings, the adversary can easily get many non-trivial encodings of zero, but only at the highest level. This difficulty getting intermediate encodings of zero makes this construction seem more secure than key agreement. The adversary also is not given an
encoding of 1 at level 1 like it is in key agreement. These parameters which are not made public in the obfuscation setting are ones which are critical in attacks against all the existing multilinear maps candidates.
Chapter 4

Attacks

This section explains the current attacks against the candidate constructions of multilinear maps. Most attacks are against the older candidates, but some variants are still unbroken. For example, against GGH13, there are attacks that make use of the encodings of 0 in the public parameters used as randomizers. However, it is not necessary to publish these in the program obfuscation case, so these attacks fail.

4.1 GGH13

The goal when attacking GGH13 is to use the public parameters to either recover the secret parameters or to find another way to break discrete log, CDH, or DDH. There are two ends of the attacks, those that compute as much as they can and those that assume the adversary has gotten some value and show how that could break the scheme. When they meet, there is a full break.

4.1.1 Attacks given encodings of 0

When we assume that the adversary is given encodings of zero at level 1, this turns out to be all that the adversary needs to recover some of the secret parameters. The intuition behind this is that the adversary has the ability to generate encodings of zero which when zero
tested are small. This means that they are not wrapped mod q. This presents a weakness.

**Computing \langle h \rangle**

The first attack recovers a basis for \langle h \rangle assuming only many random encodings of zero at the highest level. This is plausible in the key agreement case since we are given encodings of zero at level 1 and can increase the level as well as the ability to sample random elements and multiply to increase the level. In the obfuscation application, we can run the program and since there are only two possible resulting matrices, we can get many encodings of zero at the top level by subtracting results of many different input vectors with the same output.

Given many level \( k \) encodings of zero \( z_i \), we know that each is of the form \( z_i = \frac{r_i g + 0}{z^k} \).

When we multiply them by the zero testing parameter, we get

\[
\begin{align*}
z_i p_{zt} &= \frac{r_i g + 0 h z^k}{z^k g} \\
&= r_i h \quad \text{mod } q \\
&= r_i h
\end{align*}
\]

and because this was an encoding of zero, the result is small so it holds without the mod q. If we get many of these, since the encodings themselves were random and independent, we will quickly have enough so that the \( r_i h \) form a basis for \( \langle h \rangle \).

**Computing \langle h \cdot g \rangle**

This is similar except it will require lower level encodings of zero. This attack could work with low level encodings not at level 1, but since those are provided in key agreement, we assume that we are given many encodings of zero at level 1, \( x_i \). Also assume as in the key agreement case that we can sample random encodings at level 1, call them \( u_j \). Take many products of two \( x_i \) and \( k - 2 \) different \( u_j \). These will be random encodings of zero at level \( k \). Because we have many independent zeros and independent encodings of random elements,
these products, call them each \( v \), are of the form \( v = \frac{rg^2}{z^k} \).

Multiply \( v \) by the zero testing parameter

\[
v_{pt} = r g^2 h z^k \frac{z^k}{g} \equiv r g h \mod q \\
= r g h
\]

again, these results hold without reducing mod \( q \) since they are encodings of zero multiplied by the zero testing parameter. Using different \( x_i \) and \( u_j \), we have independent \( r \) for each \( v \) and if we use different \( v \), quickly we get a basis for \( \langle h \cdot g \rangle \).

**Computing \( \langle g \rangle \)**

Begin by computing the bases for \( \langle h \rangle \) and \( \langle h \cdot g \rangle \). Using \( \langle h \rangle \) we can compute a basis for \( \langle 1/h \rangle \) in the field of fractions. Using this and the basis for \( \langle h \cdot g \rangle \), we can compute an integer basis for \( \langle g \rangle \). This part assumes the same assumptions as the previous parts which are consistent with the public parameters in key agreement for GGH13.

**HJ15 Attack**

The attack of [HJ15] requires the assumption that we are given encodings of zero. This requires the key agreement setting since we do not know how to get this without the low level encodings of zero. This attack does not attempt to recover the secret parameters, but instead just break CDH without discrete log.

The attack is given a \( n \) level 1 encodings of random elements for a GGH13 scheme with zero testing at level \( n - 1 \), and the goal is to compute the result of extract on the product of the encodings at level \( k = n - 1 \). Assume that there are two public encodings of zero at level 1, \( x_1 \) and \( x_2 \) such that \( x_i = \frac{rg}{z^k} \).

Take the product of an encoding of zero, \( k - 1 \) copies of the encoding of 1, and the zero
testing parameter to get

\[ Y = y^{k-1}xp_{zt} \mod q \]
\[ = \frac{(1 + rg)^{k-1} r_i g h z^k}{z} \mod q \]
\[ = h(1 + rg)^{k-1} r_i \]

Similarly compute with two factors of zero for \( i = 1, 2 \).

\[ X_i = y^{k-2}x_i x_{1pzt} \mod q \]
\[ = h(1 + rg)^{k-2} r_1 r_2 g \]

Since we assumed that there are only two encodings of 0 given, we know that a challenge element is \( V = vy + u_1 x_1 + u_2 x_2 \) for random \( u_1 \) and \( u_2 \) because of the way sampling level 1 encodings is defined. Compute the following product

\[ W = Vy^{k-2}x_{1pzt} \mod q \]
\[ = vY + u_1 X_1 + u_2 X_2 \mod q \]
\[ = vY + u_1 X_1 + u_2 X_2 \]

Compute \( W \mod Y, X_1 \mod Y, \) and \( X_2 \mod Y \). Then we can compute \( W' \) such that \( W' - W = 0 \mod Y \) and \( W' = u'_1 X_1 + u'_2 X_2 \). Let \( v^0 \) be \( (W' - W)/Y \). This is another encoding of \( v \), however not necessarily short. This is an equivalent secret.

1. There are \( k+1 \) secrets published, \( v_1, \ldots v_{k+1} \). Generate \( k+1 \) equivalent secrets and take the product of them and call in \( \nu \). Compute \( \nu' = Y \nu \). Note that \( \nu' = Y \prod_{i=1}^{k+1} v_i + \zeta r_1 g \) where \( \zeta \in R \).

2. Let \( \nu'' = \nu' \mod X_1 \). This makes the size of the \( \nu'' \) smaller than the size of \( X_1 \). Also, note that \( \nu'' = Y \prod_{i=1}^{k+1} v_i + \zeta' r_1 g \) because \( X_1 \) is a multiple of the second term.
3. Compute $\nu'' = y(x_1)^{-1}\nu'' \mod q$.

$$
\nu'' = y(x_1)^{-1}\nu'' \mod q \\
= \frac{h(1 + rg)^K}{g} \prod_{i=1}^{k+1} v_i + \zeta''(1 + rg) \mod q
$$

Notice that the second term is small because the previous sum was of two small terms. The first term is the result of the extract procedure on the correct product, so $\nu''$ is the break of CDH. The high order bits are the shared key.

This attack could be extended to the setting where there are more than 2 encodings of zero because that would only affect finding an equivalent secret. This could still be done and only a linear increase in running time since we only need to change two of the $u_i$. This attack also requires an encoding of 1. This is important since in the last step, it needs the $k$ encodings of 1 to not affect the product of the encoded value.

4.1.2 Breaking GGH13 with a small multiple of $1/h$

If we assume that the adversary does not get an encoding of zero at level one since that would seem to break the scheme, what else seems to be enough to break it?

Suppose we could get a small multiple of $1/h$ in the fraction field. Then we could multiply with the zero testing parameter to get $\frac{d^k}{g}$ where $d$ is small. Then if we square this and multiply by an encoding of zero at level 1, we get $\frac{d^2z^{2k-1}}{g}$. Because $d$ is small, so is $d^2$. So this is a zero tester for level $2k - 1$. This allows us to break DDH because we can just extract at the higher level if we multiply all the challenge elements together. This break requires an encoding of zero at any intermediate level and a small multiple of $1/h$. This type of attack is the reason that $h$ is included in the zero testing parameter and is designed to be only somewhat small, on the order of $\sqrt{q}$. 

33
4.2 CLT14

CLT has been broken by [CHL⁺14]. The attack relies on public encodings of zero. As the adversary, we are given many encodings of zero, $x_j$, many encodings of random elements, $x'_k$, an encoding of one, $y$, the zero testing parameter $p_{zt}$ and the product of all the moduli, $x_0$.

For a given $c$ which is a level one encoding, define

$$w_{ij} = [c \cdot x'_j x_k y^{k-2} p_{zt}]_{x_0}$$

$$= \sum_{i=1}^{n} h_{i,1} [c \cdot x'_j x_k y^{k-2} z^{k} g_i^{-1} \mod p_i]_{p_i} \frac{x_0}{p_i} \mod x_0$$

$$= \sum_{i=1}^{n} x'_i j h'_i c r_{i,k} \mod x_0$$

where $h'_i = h_{i,1}(r_i g_i + 1)^{k-2} x_0 / p_i$. This is a product of the zero tester with an encoding of zero so it holds without $\mod x_0$. We can rewrite this as a matrix product when we iterate over all $j, k$.

$$W_c = \begin{pmatrix} x'_{1,1} & \cdots & x'_{n,1} \\ \vdots & \ddots & \vdots \\ x'_{1,n} & \cdots & x'_{n,n} \end{pmatrix} \begin{pmatrix} c_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & c_n \end{pmatrix} \begin{pmatrix} h'_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & h'_n \end{pmatrix} \begin{pmatrix} r_{1,1} & \cdots & r_{n,1} \\ \vdots & \ddots & \vdots \\ r_{1,n} & \cdots & r_{n,n} \end{pmatrix}$$

$$= X' \text{diag}(c_1, \ldots, c_n) \text{diag}(h'_1, \ldots, h'_n) R$$

If we choose $c$ to be $x'_1$ and then $x'_2$ for $W_1, W_2$, we get that

$$W_1 \cdot W_2^{-1} = X' \text{diag} \left( \frac{x'_{1,1}}{x'_{1,2}}, \ldots, \frac{x'_{1,1}}{x'_{1,2}} \right) X'^{-1}$$
We can compute the eigenvalues of this which will be $\frac{x'_{i,1}}{x'_{i,2}}$, however we do not get them necessarily in the same order and these fractions will be reduced, so we get $(x'_{i,1}, x'_{i,2})$. Taking the the gcd of $x'_{i,1}x'_{2} - x'_{i,2}x'_{1}$ and $x_{0}$ will be $p_{i}$ since that is a factor of both and with high probability, the other large prime factors of $x_{0}$ will not be in $x'_{i,1}x'_{2} - x'_{i,2}x'_{1}$. Once we know $p_{i}$ for all $i$, we can use our encodings of zero and encodings of one and by rational reconstruction, we can recover $g_{i}$. This is the secret parameter in CLT, so this breaks the scheme.

The attempted fixes in [BWZ14] and others still do not fix the problem presented in this attack [CLT14]. The nonlinear zero testing parameter fix in [CLT15] prevents the adversary from solving all the equations which stops this type of attack.

4.3 GGH15

This candidate construction does not have a successful attack against it, but it still has the similar problem if intermediate encodings of zero are made public. Two encodings of zero along a path is sufficient to generate a trapdoor for the matrix at the corresponding sink. This trapdoor enables a total break of the scheme.

If we are given an encoding of zero, then it is nearly a trapdoor. $CA_{u} = E \mod q$ which is small, but not quite zero. If we augment $C$ by adding rows of $-E$ to the bottom, and augmenting $A_{u}$ to have rows or the identity, then $C'A'_{u} = 0 \mod q$. This is close to a trapdoor, but it is not full rank. If we have a second zero, then doing this twice should result in a full rank trapdoor for the augmented $A'_{u}$. If we are given a challenge encoding that is with respect to a path starting at $u$, we can augment it with zeros and then $A'_{u}$ is precisely the trapdoor required. By [GPV07], we can recover the plaintext.

It is possible to recover an encoding of zero even if one is not explicitly given if the graph is improperly created. Suppose we had the graph in figure 4-1. Then we could take the top path and sample an encoding of $\alpha$. We could take the bottom path and also sample using the same subset sums to get an encoding of $\alpha$. Both have the same source and sink so we can subtract to get an encoding of zero. This demonstrates that not every DAG is secure. There must also be only at most one path between every pair of vertices.

35
4.4 Other Attacks

There have been other attacks on related problems. In GGH13, it seemed to be possible to get a basis for \( \langle g \rangle \) and other useful ideals. There has been recent work into the lattice problems for recovering the generator from a basis.

4.4.1 SVP

The first problem which has seen attention is the Shortest Vector Problem. There has been a recent algorithm for standard lattices by Regev et al [ADRS14] but this is still exponential. It is the best know algorithm for solving this problem. So if an attack were to use a shortest vector algorithm, it would seem to need to use the ideal structure of the lattice in GGH13, otherwise, the attack would be exponential. There is recent working [CL15] looking into how short a vector needs to be to break GGH13 and interesting ways to get such short vectors.

4.4.2 Bernstein

Dan Bernstein proposed an algorithm [Ber14] which attempts to solve the shortest vector problem in ideal lattices. The idea is that we can reduce the problem of shortest vector in the ideal lattice to closest vector in the log unit lattice. The log unit lattice has half
the dimension. He claims that if the field used has small prime factors, then we can use symmetries to further reduce the dimension. Though he does not provide a proof, he believes that this approach could yield a subexponential algorithm in the correct fields. This is in fact how the [CDPR15] attack finds a shortest vector in cyclotomic fields.

4.4.3 Soliloquy

The soliloquy [CGS] attack is important because it presented a polynomial time quantum algorithm for solving the shortest vector problem in a particular type of lattice.

This solves the problem by using a quantum algorithm to get a special hint which the Gentry-Szydlo algorithm [GS02] can use to solve the principal ideal problem in polynomial time. It finds the generator of the lattice multiplied by its dual through a quantum algorithm, then the rest is classical polynomial time. This attack is important because it was the first quantum attack against lattices, even though the lattices it affects are very specific. It only applies when the index of cyclotomic units is one in the unit group. This is not true in general.

4.5 Short Generator From Principal Ideal

Recently, there was an attack that solves the shortest vector problem in certain lattices [CDPR15]. This attack works in lattices which are ideal lattices that are also principal ideal lattices. Furthermore, there must be a guarantee that the generator of the lattice is a short vector in the lattice. However, this is precisely the case with GGH13 where $g$ is guaranteed to be short and the hidden lattice is a principal ideal generated by $g$. So in the case of key agreement where a basis for $g$ can be computed, we can recover the secret parameters.

The attack is in two phases. First it solves the principal ideal problem. This is a well studied problem and can be done in polynomial time with a quantum computer [Hal07] or in subexponential time with a classical computer [BF14]. This part has been done before. The new part involves taking the not necessarily small generator and using it to recover a
small generator.

The main contribution of the work is generating a basis for the log-unit lattice and proving that it is suitable for the \[ \text{Bab86} \] Bounded Distance Decoding solver. To construct the basis vectors for the log-unit lattice of a cyclotomic ring with index \( m \), enumerate the roots of unity as \( \zeta_i \). The \( i \)th basis vector is \( b_i \) is the logarithmic embedding of \( \zeta_i^{m-1} \zeta_i^{-1} \). The norm of the dual vectors is small enough to guarantee Babai’s BDD will work.

**Theorem** (CDPR15). *With the basis defined above, \( \| b^\lor \|^2 = O(m^{-1} \log^3 m) \)*

**Algorithm** Recover \( g \), given a basis of \( \langle g \rangle \) in a cyclotomic field:

1. Use a principal ideal problem solver to recover \( gu \) where \( u \) is a unit.

2. Take the logarithmic embedding of \( gu \) to get \( gu \)

3. Use Babai Bounded Distance Decoder in the log-unit lattice on \( gu \) to recover \( u \)

4. Recover \( g \) from \( gu \) and \( u \)

This breaks GGH13 assuming the existence of an intermediate zero. We saw that we can compute a basis for \( h \) in all applications. Then we can compute \( \frac{1}{h} \) in the field of fractions and by using the zero testing parameter, compute \( \frac{1}{g} \frac{1}{g} \). If we square the zero testing parameter and multiply by an intermediate zero, we have a new zero testing parameter at a level greater than \( k \). With the new zero testing parameter, we can multiply all the \( k+1 \) level one encodings together and multiply by \( h \). Then we can increase the level of the encoding to the level of our new zero testing parameter and extract. This breaks CDH. Using an intermediate level encoding of zero.

To get a full break, given multiple encodings of zero, we can construct a basis for \( \langle g \rangle \). This algorithm will recover \( g \). Given \( g \) it is possible to also recover \( z \) and so all the secret parameters can be recovered.
4.6 Current Status

4.6.1 Requirements For Attack

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Attack Requirements</th>
<th>Time</th>
<th>Break</th>
</tr>
</thead>
<tbody>
<tr>
<td>GGH13</td>
<td>One intermediate encoding of 0</td>
<td>Subexponential</td>
<td>CDH</td>
</tr>
<tr>
<td></td>
<td>Encoding of 1 and two encodings of 0 at level one</td>
<td>Polynomial</td>
<td>CDH</td>
</tr>
<tr>
<td></td>
<td>Many encodings of 0, one encoding of 1 at level one</td>
<td>Subexponential</td>
<td>Full Break</td>
</tr>
<tr>
<td>CLT14</td>
<td>Many intermediate encodings of 0</td>
<td>Polynomial</td>
<td>Full Break</td>
</tr>
<tr>
<td>GGH15</td>
<td>One intermediate encoding of 0</td>
<td>Polynomial</td>
<td>Full Break</td>
</tr>
</tbody>
</table>

4.6.2 Requirements For Construction

Minimal public parameters for each scheme and application of interest.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Public Parameters</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>GGH13 Key Agreement</td>
<td>Encodings of zero at level 1</td>
<td>Broken</td>
</tr>
<tr>
<td></td>
<td>Encoding of one at level 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Zero testing parameter</td>
<td></td>
</tr>
<tr>
<td>CLT14 Key Agreement</td>
<td>Encodings of zero at level 1</td>
<td>Broken</td>
</tr>
<tr>
<td></td>
<td>Encoding of one at level 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Zero testing parameter</td>
<td></td>
</tr>
<tr>
<td>GGH15 Key Agreement</td>
<td>Random encodings along each edge</td>
<td>Not Broken</td>
</tr>
<tr>
<td></td>
<td>Zero testing parameter</td>
<td></td>
</tr>
<tr>
<td>Program Obfuscation</td>
<td>Encodings of randomized program matrices</td>
<td>Not Broken</td>
</tr>
<tr>
<td></td>
<td>Zero testing parameter</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Encoding of identity at highest level</td>
<td></td>
</tr>
</tbody>
</table>

4.7 Open Problems

All the attacks on these multilinear maps require encodings of zero at intermediate levels. Intuitively, they all use intermediate zeros so that they can create encodings of zero at the
highest level and the zero tester leaks too much information about those. It is open to see if there are attacks against these multilinear map candidates that do not require public encodings of zero. Or at least no public encodings of zero below the zero testing level.

The next problem is attacks against the program obfuscation construction. Currently there are no known attacks against program obfuscation because it does not publish encodings of zero. Further, by using the randomizing matrices, it forces GGH13 to have the nice property of GGH15 that there is control over which order the encodings are multiplied. In GGH15 there is an explicit graph. In GGH13 with program obfuscation, the randomizing matrices are not do not commute, so there is an implicit line graph. This reduces the attacker’s ability to reuse an encoding or change the order of multiplication attempting to get an encoding of zero.

Finally, there are no proofs of security. GGH15 is close to Ring-LWE, but it is in a higher dimension. It would be nice to have a security reduction for one of these candidates. GGH15 would likely reduce to Ring-LWE, but there is not an obvious problem for GGH13. Naively one would attempt to reduce to SVP, but over the cyclotomic ring which GGH13 uses, SVP is known to be subexponential.

4.8 Conclusion

There is currently a puzzling status for candidate multilinear maps, particularly GGH13. GGH13 has a subexponential break against its key agreement protocol. However, the attack does not extend to program obfuscation. This gives the interesting result that we can get key agreement indirectly from program obfuscation but the native key agreement is broken. Program obfuscation implies key agreement yet we can construct obfuscation and our key agreement is broken.

GGH13 is broken whenever an encoding of zero is made public in any intermediate level. The attacks on CLT14 also require an encoding of zero at an intermediate level, however CLT14 has additional structure for an adversary to exploit so it should be considered weaker than GGH13.
GGH15 is new so there are not many attacks against it, but it has some interesting properties that the other schemes do not which seem to help security. It is the only candidate which does not allow a user to encode a specific element or even know what it is that he or she is encoding. And related to this, there is no need to re-randomize encodings since all sampled encodings are subset sums. This allows GGH15 to keep the encodings of zero which are trapdoors to remain secret. Another interesting property is the graph structure of GGH15. Program obfuscation actually enforces an implicit graph, but GGH15 makes the graph explicit. The levels of GGH13 are strictly weaker requirements for which encodings can be added or multiplied.

GGH13 and GGH15 both provide candidate constructions for multilinear maps which are not yet broken. There is an attack against key agreement for GGH13 which runs in polynomial time, but this does not extend beyond key agreement. Both candidates provide a construction for program obfuscation through [BR13] which does not require encodings of zero which makes all currently known attacks against them fail.
Bibliography


