A Primer on Cryptographic Multilinear Maps and Code Obfuscation

by

Kenwrick Mayo

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Combinatorics & Optimization

Waterloo, Ontario, Canada, 2015

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

The construction of cryptographic multilinear maps and a general-purpose code obfuscator were two long-standing open problems in cryptography. It has been clear for a number of years that constructions of these two primitives would yield many interesting applications. This thesis describes the Coron-Lepoint-Tibouchi candidate construction for multilinear maps, as well as new candidates for code obfuscation. We give an overview of current multilinear and obfuscation research, and present some relevant applications. We also provide some examples and warnings regarding the inefficiency of the new constructions. The presentation is self-contained and should be accessible to the novice reader.

Acknowledgements

I would like to thank my supervisor, Dr. Alfred Menezes for his excellent advice, careful reading, and words of encouragement.

I would also like to thank my readers David Jao and Doug Stinson for their scrutiny.

Finally, I would like to thank my fiance for her unwavering support.

Table of Contents

Chapter 1

Introduction

This thesis provides a broad overview of the new multilinear map and code obfuscation candidates. It is meant to be readable by any cryptographer interested in the power of these new tools. No background knowledge of multilinear maps or code obfuscation is assumed. This thesis is divided into three main chapters which can, for the most part, be read independently of each other.

1.1 Multilinear Maps

Multilinear maps (MLMs) are generalizations of bilinear pairings. The first constructive use of bilinear pairings in cryptography came in 2000 when Joux used them to construct a one-round three party Diffie-Hellman key exchange [38].

A one-round two-party Diffie-Hellman key exchange is easy using traditional assumptions about the discrete log problem: In order to establish a shared secret over an unsecured channel, Alice and Bob first agree publicly on a generator g for a finite cyclic group in which the discrete log problem is hard.

Alice chooses a secret integer a, and Bob chooses a secret integer b. They broadcast g^a and g^b respectively:

$$\begin{tikzcd}
A \arrow[r, shift left, "g^a"] & B \arrow[l, shift left, "g^b"] \\
(g^b)^a & (g^a)^b
\end{tikzcd}$$

Alice computes $(g^b)^a = g^{ab}$, and Bob computes $(g^a)^b = g^{ab}$, thus establishing a shared secret.

Joux generalized this construction to accommodate three parties. Indeed, if we have a function $e \colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}_{\mathbb{T}}$ such that $e(g^a, g^b) = e(g, g)^{ab}$, where the discrete log is hard in $\mathbb{G}$ and $\mathbb{G}_T$ (such a function is known as a cryptographic bilinear pairing), then Alice, given g^b and g^c can compute $e(g^b, g^c)^a = (e(g, g)^{bc})^a = e(g, g)^{abc}$. Similarly, Bob can compute $e(g^a, g^c)^b = (e(g, g)^{ac})^b = e(g, g)^{abc}$, and Carol can compute $e(g^a, g^b)^c = e(g, g)^{abc}$ so that the shared secret is $e(g, g)^{abc}$.

$$\begin{tikzcd}
A \arrow[rr, shift left, "g^a"] \arrow[dr, shift left, "g^a"] & & B \arrow[ll, shift left, "g^b"] \arrow[dl, shift right, "g^b"'] \\
& C \arrow[ul, shift left, "g^c"] \arrow[ur, shift right, "g^c"'] &
\end{tikzcd}$$

In fact, it is easy to see that Joux's construction generalizes. Given $e \colon \mathbb{G}^{n-1} \to \mathbb{G}_{\mathbb{T}}$ such that $e(g^{m_1}, \ldots, g^{m_{n-1}}) = e(g, \ldots, g)^{m_1 \cdots m_{n-1}}$, party i can compute the shared secret as $e(g^{m_1}, \ldots, g^{m_{i-1}}, g^{m_{i+1}}, \ldots, g^{m_n})^{m_i}$. A function with this property is called *multilinear*. It was not until 2013 that Garg, Gentry and Halevi (GGH) used ideal lattices to produce the first candidate construction of a multilinear map [29].

Current multilinear map constructions do not achieve the ideal notion of multilinear. They can only be considered *approximate* multilinear maps, because they are "noisy". Only a limited number of multilinear operations can be performed before the noise grows too large. These "approximate" mutilinear maps are known as *Graded Encoding Schemes* (GES). Despite this limitation, the GGH multilinear map candidate has spawned many new applications of multilinear maps. Examples include attribute-based encryption [33], programmable hash functions [28], and low-overhead broadcast encryption [12].

Chapter 2 details a construction of a multilinear map over the integers by Coron, Lepoint and Tibouchi (CLT) [22]. Additionally, Chapter 2 discusses the GES model for multilinear maps, and the security and efficiency of the CLT construction.

1.2 Code Obfuscation

The new application of multilinear maps that has generated the most excitement among cryptographers is *indistinguishability obfuscation* (iO) for general circuits. Using multilinear maps and fully homomorphic encryption, Garg et al. were able to construct a candidate indistinguishability obfuscator for all polynomial size circuits [31].

Indistinguishability obfuscation is a relaxation of *virtual black box* (VBB) obfuscation. Briefly, a VBB obfuscator O is an algorithm that takes as input a program P, and outputs a functionally equivalent program $P' = O(P)$ such that P' reveals nothing of P besides what can be learned from having oracle access to P. For example, one could hard code a secret k in P and then publish P'. This gives others the ability to compute the program P without learning k.

Unfortunately, due to the existence of so-called "un-obfuscatable" functions, VBB is an impossible standard to achieve [3]. iO is a concession to this impossibility result. An indistinguishability obfuscator iO is an algorithm that takes a program P and outputs a functionally equivalent program P' such that for all pairs of equal-length equivalent programs P_0 and P_1, $P_0' = iO(P_0)$ and $P_1' = iO(P_1)$ are indistinguishable. Unlike a VBB obfuscator, P_0' may leak information about P_0, as long as P_1' also reveals the same information about P_0 (since otherwise, one could use this difference to distinguish between P_1' and P_0').

iO is surprisingly powerful. At first glance, it may seem difficult to reliably hide a secret k using iO because of the potential for $iO(P)$ to directly reveal k. However, as long as one can prove the *existence* of a functionally equivalent circuit that does *not* reveal k, the indistinguishability property guarantees that $iO(P)$ also does not reveal k.

For example, Garg et al. consider the application of restricted-use software. Suppose that a software company has written a program P for which they would like to publish a demo D. D ought to be a restricted version of P that is unable to access certain features. Typically, the easiest way to implement D would be to write a wrapper L for P that specifically limits P's functionality. Since L is completely outside P, it is easy to write, and requires no modification to the code of P.

Unfortunately, $L(P)$ is easy to reverse engineer. Publishing $L(P)$ hides none of the code for P, and so savvy users could access forbidden features. Clearly, publishing $\mathrm{VBB}(L(P))$ solves this problem. It is also true that publishing $iO(L(P))$ prevents reverse engineering. This is because in principle, there exists a (suitably padded) program D that is functionally equivalent to $L(P)$ and has the same size as $L(P)$, but contains no code that implements forbidden features at all. Indeed, the software company could spend a lot of resources altering their code for P to produce D. The indistinguishability property means that $iO(D)$ is indistinguishable from $iO(L(P))$. But nothing in $iO(D)$ implements forbidden features. If one could access forbidden features in $iO(L(P))$, then it would be easy to distinguish between $iO(L(P))$ and $iO(D)$, because $iO(D)$ by definition is unable to perform forbidden operations. The mere existence of D guarantees that $iO(L(P))$ hides restricted features just as well as $iO(D)$ which hides them perfectly.

Chapter 3 presents the candidate iO construction of Garg et al. including all the necessary background. We will see exactly how multilinear maps can be leveraged to achieve iO. Chapter 3 also serves as a warning to the reader regarding the efficiency of iO. Indeed, we demonstrate that current iO constructions are hopelessly impractical, and stress that iO and its applications remain of purely theoretical interest.

1.3 Applications

The candidate iO algorithm from Garg et al. has given cryptographers unprecedented power. Applications of iO include but are not limited to functional encryption, signature schemes with fast signing time, deniable encryption, replacing random oracles with concrete hash functions, non-interactive key exchange (NIKE), and broadcast encryption.

Chapter 4 focuses on three applications of multilinear maps and code obfuscation: broadcast encryption, NIKE, and replacing the random oracle in RSA Full Domain Hash (RSA-FDH).

A broadcast encryption scheme lets a sender broadcast a message to a set G of identities, while only allowing a subset $S \subset G$ of them to decrypt the message. The set S should be specifiable at encryption time, and is therefore allowed to change for different encryptions. Broadcast encryption schemes exist without the use of multilinear maps, but using a multilinear map allows compression of the asymptotic public key size. We will see a broadcast encryption scheme where the public key size depends only logarithmically on the size of G, as opposed to the linear dependency that existing broadcast encryption schemes require [12]. Unfortunately, while the number of “group” elements in a public key is drastically reduced using a multilinear map, the size of those group elements is much larger. The result is a scheme that has larger public keys than a broadcast scheme based on bilinear pairings for all sets of users G with $|G|$ less than about 109 billion.

As an example of an application that only requires indistinguishability obfuscation and not virtual black box obfuscation, we present Boneh and Zhandry's NIKE [14]. Just as the use of multilinear maps yields a NIKE, so too does the use of an indistinguishability obfuscator. Since multilinear map constructions do not achieve the ideal notion of multilinear maps, NIKEs based only on multilinear maps require the use of a trusted third party (TTP). We will see how the additional structure of iO enables a NIKE with no TTP.

Finally we discuss the Hohenberger, Sahai, Waters (HSW) version of RSA Full Domain Hash (RSA-FDH) [37]. They make a small alteration to the traditional RSA-FDH signature scheme in order to remove the random oracle assumption that was believed to be necessary for the security proof. If N is an RSA modulus, (e, d) is an RSA public-private key pair, and $H \colon \{0,1\}^* \to \mathbb{Z}_N$ is a full domain hash function modeled as a random oracle, then an RSA-FDH signature on a message $m \in \mathbb{Z}_N$ is $\sigma = H(M)^d \mod N$. HSW replace H with a clever choice of obfuscated program.

Intuitively, obfuscated programs behave as oracles to their underlying functions, which is why it is still possible to prove security of the HSW signature scheme. Chapter 4 details exactly how the drawback of indistinguishability obfuscation (as compared to VBB) is avoided. We note that using iO to replace the random oracle model merely replaces the "untrusted" random oracle model with an even less well-studied generic model.

Chapter 2

Cryptographic Multilinear Maps

In 2003, Boneh and Silverberg published [10], establishing the utility of so-called cryptographic multilinear maps. Boneh and Silverberg described several applications for the hypothetical primitive, ranging from one-round, multi-party Diffie-Hellman key exchange to broadcast encryption. They also pointed out that constructing such a multilinear map was likely to be significantly more difficult than constructing a bilinear map. In fact, Boneh and Silverberg concluded that multilinear maps "might have to either come from outside the realm of algebraic geometry, or occur as unnatural computable maps". That is, for geometric reasons, it is unlikely that multilinear maps can be constructed in the same way as bilinear maps.

The problem of constructing a cryptographic multilinear map remained an open problem until Eurocrypt 2013, where Garg, Gentry and Halevi (GGH) presented their construction of an "approximate multilinear map" based on ideal lattices [30]. These new multilinear maps use a noise component for security, and therefore, as in fully homomorphic encryption, are limited to a fixed number of arithmetic operations before the noise grows too large. Approximate multilinear maps are referred to as Graded Encoding Schemes.

Following the GGH paper, there was a flurry of activity in the field. Several new applications of Graded Encoding Schemes were proposed, and Coron, Lepoint and Tebouchi (CLT) presented an approximate multilinear map construction that works over the integers, rather than ideal lattices [23]. Their construction is also an example of a Graded Encoding Scheme.

This chapter will explore the development of the CLT multilinear map, highlighting the new ideas that enabled its creation. The complete construction of the CLT multilinear map will be presented in the context of a one-round multi-party Diffie-Hellman key exchange, and the Graded Encoding Scheme generalization will be examined. Finally, there will be some discussion of parameter selection, efficiency, and attacks. Of particular interest is the "zero-izing" attack described in [19], which seems to eliminate the original verion of CLT as a viable multilinear map for certain applications.

2.1 CLT Construction

In this section, the CLT multilinear map construction is presented in detail. For expository purposes, this construction will be presented in the context of a one-round multi-party Diffie-Hellman key exchange. While the CLT multilinear map construction is neither the original multilinear map, nor a state-of-the-art construction, it is more intuitive due to being set over the integers. Furthermore, CLT will be a useful reference point when considering abstractions and applications.

For comparison purposes, it will be useful to recall the Boneh-Silverberg definition of a multilinear map:

Definition 2.1.1 (Multilinear Map)**.** Let $\mathbb{G}$ and $\mathbb{G}_T$ be groups of the same prime order. An n-multilinear map is a function $e\colon \mathbb{G}^n \to \mathbb{G}_T$ such that

1. $\forall x_i \in \mathbb{G}$ and $a_i \in \mathbb{Z}$, $e(x_1^{a_1}, \ldots, x_n^{a_n}) = e(x_1, \ldots, x_n)^{a_1 \cdots a_n}$.
2. If $g \in \mathbb{G}$ is a generator for $\mathbb{G}$, then $e(g, \ldots, g)$ is a generator for $\mathbb{G}_T$.

2.1.1 Motivation and Intuition

The CLT multilinear map construction is somewhat complicated. It has several components whose purpose may not immediately be clear. As such, it is helpful to start from scratch and examine the reasoning behind each design decision.

First, recall the one-round multi-party key exchange using ideal MLMs:

After a user chooses a secret integer a_0, she performs three steps:

1. Conceal the secret by computing and broadcasting g^{a_0}.
2. Compute the public data as $e(g^{a_1}, \ldots, g^{a_n})$.
3. Combine the secret key with the public data by computing $e(g^{a_1}, \ldots, g^{a_n})^{a_0}$.

Here e is a map that satisfies the multilinear property (Definition 2.1.1). The secret parameters are the a_i's. The public parameters are $e, \{g^{a_i}\}$ and g.

In the first step, $a_0 \mapsto g^{a_0}$ can be thought of as an *encoding* of a_0 in the group $\mathbb{G}$. The important property of this encoding function is that it is one-way. Users need to be able to broadcast g^{a_0} without revealing a_0. Exponentiation in $\mathbb{G}$ is a convenient, well understood one-way function (as long as the discrete log is a hard problem in $\mathbb{G}$), but could be replaced in this scheme by any other one-way function. It has proven difficult to construct a map e that satisfies the multilinear property when the encoding function is exponentiation. Perhaps a good choice of encoding function can make constructing e much easier. In particular, the simplest multilinear function is just multiplication. One might wonder if the encoding function can be chosen so that we can set e to be

$$e(x_1, \ldots, x_n) = \prod_{i=1}^{n} x_i.$$

It may be easier to take something that is already multilinear and make it secure than it is to take something that is already secure and make it multilinear.

Randomized Encoding Function

The CLT scheme replaces $a_0 \mapsto g^{a_0}$ with a randomized encoding function. Consider the encoding function

$$f : \mathbb{Z} \to \mathbb{Z}$$

given by

$$f(m) = m + r$$

where r is an integer of a fixed size chosen uniformly at random. The multi-party key exchange protocol becomes:

1. Conceal the secret by computing $f(a_0) = a_0 + r_0$.
2. Compute the public data as $e(a_1 + r_1, \ldots, a_n + r_n) = \prod_{i=1}^{n}(r_i + a_i) = R + \prod_{i=1}^{n} a_i$.
3. Combine the secret key with the public data by computing $e(f(a_1), \ldots, f(a_n)) \cdot a_0 = R' + \prod_{i=0}^{n} a_i$.

The secret parameters are the a_i's and r_i's. The public parameters are the $f(a_i)$'s.

This setup has a problem. The noise R' will be different for each user. There must be a way to extract the deterministic signal (namely $\prod_{i=0}^{n} a_i$) from

$$a_0 \prod_{i=1}^{n} f(a_i).$$

However, an attacker should not be able to extract a signal from

$$\prod_{i=0}^{n} f(a_i).$$

In fact, the extraction function should only work for exactly a product of n public keys and one private key, and no other partial or extraneous product.

Therefore there must be a way for the extraction function to determine how many public and private key factors its input has. This is analogous in the Boneh-Silverberg definition to needing $\mathbb{G}$ and $\mathbb{G}_T$ to be different and distinguishable.

Distinguishing inputs from outputs

This can be accomplished by modifying the simple encoding function given above. Instead of $f(m) = m + r$, consider

$$f : \mathbb{Z}_p \to \mathbb{Z}_p$$

for some prime p, given by

$$f(m) = \frac{r+m}{z} \mod p$$

where r is again random, and z is a fixed integer. The multi-party key exchange protocol becomes:

1. Conceal the secret by computing $f(a_0) = \frac{r_0+a_0}{z} \mod p$.
2. Compute the public data as $e(f(a_1), \ldots, f(a_n)) = \prod_{i=1}^{n} \frac{r_i+a_i}{z} = \frac{R+\prod_{i=1}^{n} a_i}{z^n} \mod p$.
3. Combine the secret key with the public data by computing

$$e(f(a_1), \ldots, f(a_n)) \cdot a_0 = \frac{R' + \prod_{i=0}^{n} a_i}{z^n} \mod p.$$

The secret parameters are the a_i's and the r_i's. z must also be kept secret, but doing so presents some challenges. The secrecy of z will be discussed shortly. The public parameters are p and the $f(a_i)$'s.

The exponent of the denominator indicates the number of public keys that have been multiplied together. This facilitates the design of an extraction function that only works for a product of a specific number of public keys. All that is left to do is specify the extraction function.

Extracting Deterministic Information

In order to cancel the randomness, another modification of the encoding function is necessary. Consider $f\colon \mathbb{Z}_g \to \mathbb{Z}_p$ given by

$$f(m) = \frac{rg+m}{z} \mod p,$$

where g is a fixed prime much smaller than p, r is random and small relative to p, and z is a fixed integer. Let

$$p_{zt} = z^n g^{-1} \mod p,$$

where $n+1$ is the number of parties. Call this the *zero testing parameter.* Denote

$$f_k(m) = \frac{rg+m}{z^k} \mod p.$$

We say that $f_k(m)$ is a *level-k encoding* of m. When $m = 0$,

$$f_n(m) \cdot p_{zt} = r$$

which is small relative to p. However, if $m \neq 0$, then

$$f_n(m) \cdot p_{zt} = r + g^{-1}m \mod p$$

is similar in size to p. This enables users to test if two level-n encodings are encodings of the same element. That is, if $m = m'$, then

$$f_n(m) - f_n(m') = \frac{r_1 g + m}{z^n} - \frac{r_2 g + m'}{z^n} = \frac{Rg}{z^n} \mod p$$

which is a level-n encoding of 0, whence

$$\left(\frac{Rg}{z^n} \mod p\right) \cdot p_{zt} = R$$

which is small relative to p. Similarly, if $m \neq m'$, then

$$f_n(m) - f_n(m') = \frac{Rg + (m - m')}{z^n} \mod p$$

so that

$$(f_n(m) - f_n(m')) \cdot p_{zt} = R + g^{-1}(m - m') \mod p$$

which is of similar size to p.

In particular, $m = m'$ if and only if $f_n(m) \cdot p_{zt}$ and $f_n(m') \cdot p_{zt}$ share the first several most significant bits. These shared bits will play the role of the shared secret.

So, the final Diffie-Hellman procedure is as follows:

1. Conceal the secret by computing $f(a_0) = \frac{r_0 g + a_0}{z} \mod p$.
2. Compute the public data as $e(f(a_1), \ldots, f(a_n)) = \prod_{i=1}^{n} \frac{r_i g + a_i}{z} \mod p = \frac{Rg + \prod_{i=1}^{n} a_i}{z^n} \mod p$.
3. Combine the secret key with the public data by computing
$$a_0 \cdot e(f(a_1), \ldots, f(a_n)) = \frac{R'g + \prod_{i=0}^{n} a_i}{z^n} \mod p.$$
4. Extract the shared secret by computing the ν most significant bits of $p_{zt} \frac{R'g + \prod_{i=0}^{n} a_i}{z^n} \mod p$.

ν is a parameter that will be specified based on the security level and the multilinearity level (see Section 2.3.2).

The secret parameters are the a_i's and the r_i's. The public parameters are the $f(a_i)$'s, p, ν, and p_{zt}. z and g are also be kept secret (see Remark 2.1.2).

Secrecy of z

At the end of step 3, each user will have computed a different level-n encoding of the product of all the secret keys. Since each of these encodings is an encoding of the same message, the zero testing parameter p_{zt} allows each user to extract the same most significant bits, as described above. So, anyone who can compute a level-n encoding of $\prod_{i=0}^{n} a_i$ can compute the shared secret.

This means that the parameter z must be secret. Otherwise, an attacker can compute a level-$(n+1)$ encoding of $\prod_{i=0}^{n} a_i$, namely

$$\prod_{i=0}^{n} \frac{r_i g + a_i}{z} \mod p = \frac{R'g + \prod_{i=0}^{n} a_i}{z^{n+1}} \mod p.$$

If an attacker knows z, they can multiply their level-$(n+1)$ encoding by z to get a level-n encoding, and thus compute the shared secret.

Composite Modulus

Due to a technique called rational reconstruction, the modulus p must also be secret. Rational reconstruction is an algorithm based on the Euclidean algorithm that allows one to efficiently recover a and b from $\frac{a}{b} \mod p$ and p provided a and b are sufficiently small. If an attacker knows p, they can compute the ratio modulo p of two public keys:

$$\frac{r_0 g + a_0}{r_1 g + a_1} \mod p$$

and, since $r_0 g + a_0$ and $r_1 g + a_1$ are necessarily small relative to p, they can use rational reconstruction to recover $r_0 g + a_0$ and $r_1 g + a_1$. From here, all the attacker needs to do to recover z is compute

$$\frac{r_0 g + a_0}{z}(r_0 g + a_0)^{-1} \mod p = z^{-1} \mod p.$$

The fact that p needs to be kept secret causes a problem. Users need to be able to compute products modulo p, which is very difficult without knowledge of p. This problem can be solved by using a composite modulus and the Chinese Remainder Theorem. The full solution is described in the next section.

Remark 2.1.2 (TTP requirement)**.** The fact that z needs to be secret causes a much more fundamental problem. In order to use this scheme for agreeing on a shared secret, users must have already agreed on a shared secret z! Currently, the best “solution” to this problem is to use a trusted third party to generate the protocol parameters. Because of the reliance on a TTP, CLT multilinear maps do not yield a true Diffie-Hellman key exchange.

2.1.2 Construction of CLT

The previous section introduced the main ideas that enable the construction of a cryptographic multilinear map. However, there are several issues that remain in the described construction. This section will provide a detailed description of the full CLT construction, including the modifications that need to be made in order to keep z and p secret while still maintaining some level of functionality.

A one-round multi-party key exchange has four steps:

1. Instance Generation
2. Secret Key Generation
3. Public Key Generation and Broadcast
4. Shared Secret Computation

For example, in a standard Diffie-Hellman protocol, instance generation is selecting a group and generator g, secret key generation is selecting a random integer a, public key generation and broadcast is computing and broadcasting g^a, and shared secret computation is computing $(g^b)^a$.

A CLT Diffie-Hellman key exchange uses the same steps, and it is therefore convenient to examine each of them individually.

Preliminaries

Throughout this section, all integers expressed as reductions modulo x_0 are considered to be in the range $(-x_0/2, x_0/2)$, instead of the usual $[0, x_0)$. This is necessary for the proof of Lemma 2.1.6.

Definition 2.1.3 (level-k encoding)**.** Let $\{p_i\}$ and $\{g_i\}$ be two sets of n primes where p_i is η bits and g_i is α bits. Let $x_0 = \prod_{i=1}^{n} p_i$. A *message* is a vector $\mathbf{m} \in \mathbb{Z}_{g_1} \times \cdots \times \mathbb{Z}_{g_n}$. A *level-k encoding* of a message $\mathbf{m}$ is an integer $c \in \mathbb{Z}_{x_0}$ such that:

$$c \equiv \frac{r_i g_i + m_i}{z^k} \mod p_i$$

for all i from 1 to n where r_i is a ρ-bit random number, $m_i \in \mathbb{Z}_{g_i}$, and $z \in \mathbb{Z}_{x_0}^*$ is fixed.

Notice that the modulus x_0 can be made public without exposing encodings to rational reconstruction, since the value of $c \bmod p_i$ is kept secret. Obviously, for this to be effective, the individual p_i's must remain secret.

Definition 2.1.4. Let $\mathbf{H}$ be an $n \times n$ integer matrix. Then

$$\|\mathbf{H}\|_\infty = \underset{1\leq i\leq n}{max} \sum_{j=1}^{n} |\mathbf{H}_{ij}|.$$

Instance Generation

Given a security level λ and a multilinearity level κ, a trusted third party (TTP) does the following:

- Using λ and κ, selects appropriate values for n, η, α, and ρ.
- Randomly selects n η-bit primes $\{p_i\}_{i=1}^n$.
- Randomly selects n α-bit primes $\{g_i\}_{i=1}^n$.
- Computes $x_0 = \prod_{i=1}^n p_i$.
- Randomly selects $z \in \mathbb{Z}_{x_0}^*$.

The p_i's, g_i's, and z will all be kept secret by the TTP. The TTP then publishes the following information:

- x_0, n, η, α.
- Y, a randomly chosen level-1 encoding of the all-ones vector $\mathbf{1}$.
- p_{zt}, an n-dimensional vector with $(p_{zt})_j = \sum_{i=1}^n h_{ij}(z^\kappa g_i^{-1} \bmod p_i)\frac{x_0}{p_i} \bmod x_0$. The h_{ij}'s are the entries of an invertible $n \times n$ integer matrix $\mathbf{H}$ that satisfies:
  - $\|\mathbf{H}\|_\infty \leq 2^\beta$
  - $\|(\mathbf{H}^{-1})^T\|_\infty \leq 2^\beta$

  for a suitable choice of β.
- $\left\{x_j'\right\}_{j=1}^{\ell}$, where each x_j' is a level-0 encoding of a random message.

- n level-1 encodings of the all-zeros vector $\mathbf{0}$, $\{\Pi_j\}$.
- τ level-1 encodings of $\mathbf{0}$, $\{x_j\}$.

Appropriate values for n, η, α, β, ρ, ℓ and τ will be discussed in Section 2.3.2.

Remark 2.1.5 (Secret g_i's)**.** To the best of my knowledge, there is currently no attack that results from letting the g_i's be public. Cheon et al. allude to this possibility in Section 2.1 of [19]. However, in the original lattice based Graded Encoding Scheme by GGH, it was critical that the analog of the g_i's be kept secret (see Section 6.3.3 of [29] for details). It is reasonable to keep the g_i's secret in the CLT GES, for fear of a similar attack.

Secret Key Generation

A user's secret key is a level-0 encoding of some message $\mathbf{m}$. User k's secret key is some integer c_k such that

$$c_k \equiv r_i g_i + m_i \mod p_i$$

for all i from 1 to n, where r_i is a ρ-bit random integer. Users need to be able to generate secret keys without having access to the secret parameters p_i and g_i. This is the reason for publishing $\{x'_j\}_{j=1}^{\ell}$. Recall that each x'_j is a level-0 encoding of a random message $\mathbf{a_j}$:

$$x'_j \equiv r'_{ij} g_i + a_{ij} \mod p_i$$

for all i. Then any linear combination of the x'_j's is also a level-0 encoding of some message. In particular, a user generates a secret key by randomly selecting $b \in \{0,1\}^{\ell}$, and computing

$$\begin{aligned} c_k &= \sum_{j=1}^{\ell} b_j x'_j \mod x_0 \\ &\equiv g_i \sum_{j=1}^{\ell} b_j r'_{ij} + \sum_{j=1}^{\ell} b_j a_{ij} \mod p_i \end{aligned}$$

which is a level-0 encoding of $\sum_{j=1}^{\ell} b_j \mathbf{a_j}$. Notice that each noise component in each c_k is roughly a $\log_2(\ell) + \rho$ bit integer. This fact is useful when computing bounds of the noise of a product of public keys. See Section 2.3.2 for further details.

CLT use a modified leftover hash lemma to prove that the distribution of the messages encoded by these subset sums of level-0 encodings is statistically close to uniform.

Public Key Generation and Broadcast

If a user's secret key is a level-0 encoding of $\mathbf{m}$, then their public key is a level-1 encoding of $\mathbf{m}$. Again, it is difficult to generate such an encoding without knowing the parameters of the scheme. Recall that Y is a public level-1 encoding of $\mathbf{1}$. If c_k is a level-0 encoding of $\mathbf{m}$, then Yc_k is a level-1 encoding of $\mathbf{m} \cdot \mathbf{1}$ (where the multiplication is taken componentwise).

However, Yc_k is not a good public key, since an attacker can divide Yc_k by Y to recover c_k. The public key really ought to be a level-1 encoding of $\mathbf{m}$ selected randomly from all possible level-1 encodings of $\mathbf{m}$. To achieve this, the protocol provides a way to "re-randomize" level-1 encodings. The re-randomization procedure takes a level-1 encoding of a message $\mathbf{m}$ and returns a different, random level-1 encoding of $\mathbf{m}$.

Notice that adding any level-1 encoding of $\mathbf{0}$ to Yc_k yields another level-1 encoding of $\mathbf{m}$. Recall that the TTP published two sets of level-1 encodings of $\mathbf{0}$, namely $\{\Pi_j\}$ and $\{x_j\}$. Similar to the secret key generation, a user can re-randomize Yc_k by adding a subset sum of the x_i's:

$$c_k' = Yc_k + \sum_{j=1}^{\tau} b_j x_j \mod x_0$$

where again $\mathbf{b}$ is a random vector in $\{0,1\}^\tau$. Unlike in the secret key generation, this is not enough to ensure that c_k' is independent from Yc_k. CLT's proof of (near) independence requires that the user computes

$$c_k' = Yc_k + \sum_{j=1}^{\tau} b_j x_j + \sum_{j=1}^{n} b_j' \Pi_j \mod x_0.$$

Here, each b_j' is a random μ-bit integer. There are also some technical conditions on the structure of $\{x_j\}$ and $\{\Pi_j\}$ that are necessary for their proof to follow; for details, see [22]. Using these conditions, CLT calculate that each noise component of c_k' is less than $\ell 2^{2\rho+\alpha} + \tau n 2^{\rho+1} + n^2 2^{\mu+\rho+1}$. Again, this will be useful when selecting parameter sizes in Section 2.3.2.

Shared Secret Computation

Recall that the TTP published the n-dimensional vector p_{zt} with

$$(p_{zt})_j = \sum_{i=1}^{n} h_{ij} (z^\kappa g_i^{-1} \bmod p_i) \frac{x_0}{p_i} \mod x_0.$$

Here the h_{ij}'s are the entries of an invertible $n \times n$ integer matrix $\mathbf{H}$ such that:

- $\|\mathbf{H}\|_\infty \leq 2^\beta$
- $\|(\mathbf{H}^{-1})^T\|_\infty \leq 2^\beta$.

p_{zt} is called the *zero-testing parameter*. It can be used to check if a level-κ encoding is an encoding of the message $\mathbf{0}$ or not. Specifically, the following lemma holds:

Lemma 2.1.6 (Zero-Testing). *Let ρ_f be an integer such that $\rho_f + \lambda + \alpha + 2\beta \leq \eta - 8$, and let $\nu = \eta - \beta - \rho_f - \lambda - 3 \geq \alpha + \beta + 5$. Let c be a level-κ encoding of a message $\mathbf{m}$, so that $c \equiv \frac{r_i g_i + m_i}{z^\kappa} \mod p_i$ for all $1 \leq i \leq n$. Assume $\|\mathbf{r}\|_\infty \leq 2^{\rho_f}$. If $\mathbf{m} = \mathbf{0}$ then $\|cp_{zt} \mod x_0\|_\infty < x_0 2^{-\nu-\lambda-2}$. Conversely, if $\mathbf{m} \neq \mathbf{0}$ then $\|cp_{zt}\|_\infty \geq x_0 2^{-\nu+2}$.*

In particular, if c_1 and c_2 are different level-κ encodings of the same message $\mathbf{m}$, then $c_1 - c_2$ is a level-κ encoding of $\mathbf{0}$, so by Lemma 2.1.6,

$$\|(c_1 - c_2)p_{zt} \mod x_0\|_\infty < x_0 2^{-\nu-\lambda-2},$$

whence for all j,

$$|(c_1 - c_2) \cdot (p_{zt})_j \mod x_0| < x_0 2^{-\nu-\lambda-2}.$$

Thus,

$$|c_1(p_{zt})_j \bmod x_0 - c_2(p_{zt})_j \bmod x_0| < x_0 2^{-\nu-\lambda-2}.$$

In other words, $c_1(p_{zt})_j \mod x_0$ and $c_2(p_{zt})_j \mod x_0$ have the same ν most significant bits. Then in order for several parties to compute a shared secret, it suffices for them to each compute a level-κ encoding of the same message.

Each public key c'_k is a level-1 encoding of some message $\mathbf{m_k}$. Each corresponding secret key c_k is a level-0 encoding of the same message $\mathbf{m_k}$. So if there are $\kappa + 1$ users, user k computes the product

$$s_k = c_k \prod_{i \neq k} c'_i \mod x_0,$$

which is a level-κ encoding of $\prod_{i=1}^{\kappa+1} \mathbf{m_i}$. Then the ν most significant bits of each component of $s_k p_{zt} \mod x_0$ are the same for all k.

2.2 Graded Encoding Schemes

Multilinear maps were defined by Boneh and Silverberg in terms of groups. In particular, they considered two groups, an input group $\mathbb{G}$ and a target group $\mathbb{G}_T$. They define the multilinear map as a transformation between several copies of the input group and one copy of the target group that is multilinear with respect to the two group operations. This is a natural extension of the setting used in bilinear maps. Before there were candidates for instantiating multilinear maps, applications for cryptographic multilinear maps were designed with this definition in mind.

Unfortunately, the CLT setting is not an instantiation of this definition. Many theoretical protocols that used the traditional definition of a multilinear map do not work in the CLT setting. Even a simple Diffie-Hellman key exchange is substantially different in the CLT setting as evidenced by the requirement for a TTP.

In fact, none of the recent multilinear map candidates actually satisfy the definition of a multilinear map. Instead, they are all examples of so called "Graded Encoding Schemes" (GES). This section focuses on the abstract notion of a Graded Encoding scheme, and the differences between Graded Encoding Schemes and true multilinear maps.

2.2.1 Definitions

Definition 2.2.1 (κ-Graded Encoding System [GGH]). A κ-*Graded Encoding System* is a ring R and a family of sets $\mathbb{S} = \{S_k^\alpha \subset \{0,1\}^* | \alpha \in R,\ 0 \leq k \leq \kappa\}$ such that

1. for every fixed level k, the sets $\{S_k^\alpha | \alpha \in R\}$ are disjoint;
2. there are binary operations $+$ and $-$ on $\{0,1\}^*$ such that for all $\alpha_1, \alpha_2 \in R$, all k, and every $u_1 \in S_k^{\alpha_1}, u_2 \in S_k^{\alpha_2}$, we have $u_1 + u_2 \in S_k^{\alpha_1+\alpha_2}$ and $u_1 - u_2 \in S_k^{\alpha_1-\alpha_2}$;
3. there is an associative binary operation $\times$ on $\{0,1\}^*$ such that for every $\alpha_1, \alpha_2 \in R$, every k_1, k_2 with $0 \leq k_1 + k_2 \leq \kappa$, and every $u_1 \in S_{k_1}^{\alpha_1}$, $u_2 \in S_{k_2}^{\alpha_2}$, we have $u_1 \times u_2 \in S_{k_1+k_2}^{\alpha_1 \cdot \alpha_2}$.

Remark 2.2.2 (CLT is a Graded Encoding System). It is not hard to see that a fixed CLT parameter set yields a Graded Encoding System. Let $R = \mathbb{Z}_{g_1} \times \cdots \times \mathbb{Z}_{g_n}$. Identify $\{0,1\}^{\log_2 x_0}$ with $\mathbb{Z}_{x_0}$. Then for any $\mathbf{m} \in R$, let $S_k^{\mathbf{m}}$ be the set of level-k encodings of $\mathbf{m}$.

The first property in the definition requires each encoding to be an encoding of exactly one message. In CLT, this is the case, because every integer modulo p_i can be expressed uniquely in the form $\frac{r_i g_i + m_i}{z^k}$ mod p_i due to the Euclidean Algorithm.

The second and third properties correspond to the additive and multiplicative homomorphism of the CLT encodings. If $\mathbf{m_1}, \mathbf{m_2} \in R$, and $u_1 \in S_k^{\mathbf{m_1}}, u_2 \in S_k^{\mathbf{m_2}}$, then

$$u_1 + u_2 \equiv \frac{r_{i,1}g_i + m_{i,1}}{z^k} + \frac{r_{i,2}g_i + m_{i,2}}{z^k} \equiv \frac{(r_{i,1} + r_{i,2})g_i + (m_{i,1} + m_{i,2})}{z^k} \mod p_i \in S_k^{\mathbf{m_1}+\mathbf{m_2}}.$$

Similarly, if $u_1 \in S_{k_1}^{\mathbf{m_1}}, u_2 \in S_{k_2}^{\mathbf{m_2}}$, then

$$u_1 \times u_2 \equiv \frac{r_{i,1}g_i + m_{i,1}}{z^{k_1}} \times \frac{r_{i,2}g_i + m_{i,2}}{z^{k_2}} = \frac{r_i' g_i + m_{i,1}m_{i,2}}{z^{k_1+k_2}} \mod p_i \in S_{k_1+k_2}^{\mathbf{m_1 m_2}}.$$

In addition to the above definition, it is necessary to specify how users are able to interact with encodings. For example, a graded encoding system where adding encodings is an inefficient operation might not be useful. The multilinear map literature refers to Graded Encoding Schemes (GES) as Graded Encoding Systems where a certain set of procedures are efficient:

- Instance Generation: $InstGen(\lambda, \kappa)$ outputs $params$, the parameters for a κ-Graded Encoding System, and p_{zt}, a zero-testing parameter for level-κ.
- Sampler: *samp*(*params*) outputs $a \in S_0^{\alpha}$, where α is nearly uniformly random.
- Encoding: *enc*(*params*, i, a) for $i \leq \kappa$ and $a \in S_0^{\alpha}$, outputs $u \in S_i^{\alpha}$.
- Re-Randomization: *reRand*(*params*, i, u) for $i \leq \kappa$ and $u \in S_i^{\alpha}$, outputs $u' \in S_i^{\alpha}$ such that u' is distributed nearly uniformly in S_i^{α}.
- Addition, Negation: The addition and negation operations of the Graded Encoding System.
- Multiplication: The multiplication operation of the Graded Encoding System.
- Zero-test: *isZero*(*params*, p_{zt}, u) outputs 1 if $u \in S_\kappa^0$ and 0 otherwise.
- Extraction: *ext*(*params*, p_{zt}, u) outputs a λ-bit nearly uniform random string. The output of *ext* should be the same for all u that are level-κ encodings of the same message.

Remark 2.2.3 (Procedures for different applications)**.** Graded Encoding Schemes were originally designed to satisfy the use-case of a Diffie-Hellman key exchange, where users are required to generate their own public and private keys. Any application which does not require users to generate their own encodings does not need the *samp* or *reRand* procedures. Given that *samp* and *reRand* can be expensive in practice, it is important to note when they are unnecessary. It also may be the case in the future that a Graded Encoding Scheme is invented where sampling and re-randomization are not feasible. Such a GES could potentially still be useful.

Remark 2.2.4 (Comparison with multilinear maps)**.** The collection of sets $\{S_1^\alpha | \alpha \in R\}$ in a Graded Encoding System corresponds to the input group $\mathbb{G}$ of a multilinear map. Similarly, the collection of sets $\{S_\kappa^\alpha | \alpha \in R\}$ corresponds to the target group $\mathbb{G}_T$ in a multilinear map. Notice that a Graded Encoding System has some additional structure with no analog in multilinear maps, namely the sets S_j^α for $1 < j < \kappa$. These intermediate sets facilitate pairing fewer than κ input elements. A traditional multilinear map has no way to compute a meaningful pairing of fewer than κ elements. A single instance of a Graded Encoding System can therefore yield k-linear maps for all $k \leq \kappa$, where one would otherwise need κ separate multilinear maps.

This can have an impact on tasks such as multiplication. Graded Encoding Schemes have the nice property that an encoding of a product is equal to the product of encodings of its factors. Multilinear maps have a similar property, namely that an encoding of a product is equal to the *pairing* of encodings of its factors. The difference is that the product operation in Graded Encoding Schemes can be computed incrementally as a binary operation, while the pairing operation in the multilinear map setting must pair each of the factors simultaneously.

2.2.2 Asymmetric Graded Encoding Schemes

Cryptographic bilinear maps are often *asymmetrical.* The difference between an asymmetrical bilinear map and a symmetrical one is that the asymmetric map has two distinct input groups $\mathbb{G}_1$ and $\mathbb{G}_2$. This generalization can be applied to multilinear maps as well:

Definition 2.2.5 (Asymmetric Multilinear Map)**.** Let $\mathbb{G}_1, \ldots, \mathbb{G}_n$, and $\mathbb{G}_T$ be groups of the same prime order where the discrete log problem is hard. An *n-multilinear map* is a function $e \colon \mathbb{G}_1 \times \cdots \times \mathbb{G}_n \to \mathbb{G}_T$ such that

1. $\forall x_i \in \mathbb{G}_i$ and $a_i \in \mathbb{Z}$, $e(x_1^{a_1}, \ldots, x_n^{a_n}) = e(x_1, \ldots, x_n)^{a_1 \ldots a_n}$;
2. if each $g_i \in \mathbb{G}_i$ is a generator for $\mathbb{G}_i$, then $e(g_1, \ldots, g_n)$ is a generator for $\mathbb{G}_T$.

An asymmetric version of a multilinear map is useful for applications that want to further restrict the ways in which the elements can be paired. For example, an asymmetric multilinear map does not allow an attacker to pair a group element with itself.

Graded Encoding Schemes can also be modified to support asymmetric behaviour by indexing the sets S with vectors rather than integers:

Definition 2.2.6 (T-Graded Encoding System [29])**.** Let $T \in \mathbb{N}^t$ for some integer $t > 0$, and let R be a ring. A vector $\mathbf{v} \in \mathbb{N}^t$ is *below* T if $v_i \leq T_i$ for all $i \leq t$. A T-*Graded Encoding System* for R is a family of sets $\mathbb{S} = \{S_{\mathbf{v}}^{\alpha} \subset \{0,1\}^* | \alpha \in R$, and $\mathbf{v}$ is below $T\}$ such that

1. for every fixed level $\mathbf{v}$ below T, the sets $\{S_{\mathbf{v}}^{\alpha} | \alpha \in R\}$ are disjoint;
2. there are binary operations $+$ and $-$ on $\{0,1\}^*$ such that for all $\alpha_1, \alpha_2 \in R$, all $\mathbf{v}$ below T, and every $u_1 \in S_{\mathbf{v}}^{\alpha_1}, u_2 \in S_{\mathbf{v}}^{\alpha_2}$, we have $u_1 + u_2 \in S_{\mathbf{v}}^{\alpha_1+\alpha_2}$ and $u_1 - u_2 \in S_{\mathbf{v}}^{\alpha_1-\alpha_2}$;
3. there is an associative binary operation $\times$ on $\{0,1\}^*$ such that for every $\alpha_1, \alpha_2 \in R$, every $\mathbf{v_1}$ and $\mathbf{v_2}$ below T with $\mathbf{v_1} + \mathbf{v_2}$ below T, and every $u_1 \in S_{\mathbf{v_1}}^{\alpha_1}$, $u_2 \in S_{\mathbf{v_2}}^{\alpha_2}$, we have $u_1 \times u_2 \in S_{\mathbf{v_1}+\mathbf{v_2}}^{\alpha_1 \cdot \alpha_2}$, where $\mathbf{v_1} + \mathbf{v_2}$ is vector addition in $\mathbb{N}^t$.

A T-Graded Encoding Scheme is a T-Graded Encoding System equipped with the same efficient procedures as the symmetric case.

Example 2.2.7 (CLT as a T-Graded Encoding System)**.** To use CLT as a T-Graded Encoding System, instead of only generating a single secret mask z, one chooses t secret masks $\{z_i\}_{i=1}^t$. Then for any vector I that is below T, a level-I encoding of a message $\mathbf{m}$ is an integer c such that

$$c \equiv \frac{r_i g_i + m_i}{z_1^{I_1} \cdot \cdots \cdot z_t^{I_t}} \mod p_i$$

for all i. For example, a level-$(1, 2, 0, \ldots, 0)$ encoding of $\mathbf{m}$ is an integer c such that

$$c \equiv \frac{r_i g_i + m_i}{z_1 \cdot z_2^2} \mod p_i$$

for all i.

The zero testing parameter now must be defined relative to the vector T instead of relative to the integer κ:

$$(p_{zt})_j = \sum_{i=1}^{n} h_{ij} (z_1^{T_1} \cdot \cdots \cdot z_t^{T_t} g_i^{-1} \bmod p_i) \frac{x_0}{p_i} \mod x_0.$$

We will see in Chapter 3 that asymmetric Graded Encoding Schemes are used extensively in new code obfuscation techniques to ensure that only very specific products can be zero tested.

2.3 Cryptanalysis of CLT

The CLT scheme was given with no reduction to a well-studied hard problem such as approximate GCD or Learning With Errors (LWE). Indeed, the existence of such a reduction seems unlikely [29]. Instead, the authors give a new hard problem called the Graded Decisional Diffie-Hellman problem, and present a preliminary cryptanalysis of their scheme, outlining several possible avenues of attack, and demonstrating that each of the attacks takes time exponential in the security parameter λ. These initial attacks are used as motivation for parameter choices in Section 2.3.2.

The first external cryptanalysis of CLT, [41] found an attack that exploits the scheme's lack of public key validation. This attack is presented briefly in Section 2.3.3, and a heuristic method for performing public key validation is given.

Later, a more sophisticated attack based on generating high-level encodings of **0** was found by Cheon et al. [19]. Given level-1 encodings of **0**, Cheon's attack is able to find all the secret parameters of CLT, breaking CLT for any application that publishes level-1 encodings of 0 (multi-party Diffie-Hellman for example). There have been several attempts to fix CLT's vulnerability to this so called "zero-izing" attack, some of which have failed. The state-of-the-art version of CLT [25] seems to avoid the Cheon attack. The zero-izing attack and the ensuing countermeasures are documented in Sections 2.3.4 and 2.3.5 respectively.

2.3.1 Security Assumptions

The main security assumption needed for multi-party key exchange using CLT is a variant of the traditional decisional Diffie-Hellman problem called the Graded Decisional Diffie-Hellman (GDDH) problem. The GDDH assumption is that given a multilinearity κ, and a security parameter λ, no polynomial time adversary $\mathcal{A}$ has non-negligible advantage in the following security game:

1. The challenger generates the parameters for a GES, including the zero-tester p_{zt} and all the encodings necessary for sampling and re-randomization.
2. For $1 \leq i \leq \kappa + 1$, the challenger sets $a_i = samp(params)$.
3. The challenger then sets $u_i = reRand(params, 1, enc(params, 1, a_i))$ so that u_i is a random level-1 encoding of the same message encoded by a_i.

4. The challenger selects a random level-0 encoding $r = samp(params)$.
5. The challenger selects $b \in_R \{0,1\}$, and gives $\mathcal{A}$ the u_i and

$$v_b = \begin{cases} reRand(params, \kappa, enc(params, \kappa, \prod_{i=1}^{\kappa+1} a_i)), & b = 1, \\ reRand(params, \kappa, enc(params, \kappa, r)), & b = 0. \end{cases}.$$

6. $\mathcal{A}$ returns $b' \in \{0,1\}$.

$\mathcal{A}$'s advantage is $|P(b' = 1 \mid b = 1) - P(b' = 1 \mid b = 0)|$.

2.3.2 Setting Parameters

The instance generation procedure for a multilinear Diffie-Hellman key exchange needs to be able to choose concrete values for numerous parameters based on the security level λ, and the desired multilinearity level κ. Parameter sizes are chosen either to ensure the correctness of the scheme, or to avoid attacks.

Correctness

The following list of parameters must satisfy the given bounds in order for the correctness of the scheme to follow:

ℓ is the number of public level-0 encodings $\left\{x'_j\right\}$ published by the TTP. A condition to proving that a message encoded by a subset-sum of the x_j's is uniformly distributed among possible messages is that

$$\ell \geq n\alpha + 2\lambda.$$

τ is the number of public level-1 encodings $\{x_j\}$ of $\mathbf{0}$ published by the TTP. In order to prove that re-randomization produces a uniformly random encoding of the same message, it is necessary that

$$\tau \geq n(\rho + \log_2(2n)) + 2\lambda.$$

μ is the bit size of the coefficients of the Π_j's used in the re-randomization process. Again, in order for re-randomization to behave uniformly, it is necessary that

$$\mu \geq \alpha + \rho + \lambda.$$

Finally, recall that η is the bit-size of the primes p_i. A condition of Lemma 2.1.6, which guarantees the functionality of the zero-testing parameter, is that

$$\eta \geq \rho_f + \alpha + 2\beta + \lambda + 8.$$

Lemma 2.1.6 also requires that ν, the number of most significant bits to extract from the product of a level-κ encoding and the zero tester, satisfy

$$\nu = \eta - \beta - \rho_f - \lambda - 3.$$

ρ_f is the maximum of any noise component in an encoding that is to be zero-tested. For a Diffie-Hellman key exchange, ρ_f can be chosen to be the number of bits in a noise component of a product of κ public keys and a single private key. Note that for any two level-1 encodings $c_1 \equiv \frac{r_{i,1}g_i+m_{i,1}}{z} \bmod p_i$ and $c_2 \equiv \frac{r_{i,2}g_i+m_{i,2}}{z} \bmod p_i$, if $r_{i,1}$ and $r_{i,2}$ are bounded by 2^ρ, then

$$\frac{r_{i,1}g_i + m_{i,1}}{z} \cdot \frac{r_{i,2}g_i + m_{i,2}}{z} = \frac{(r_{i,1}r_{i,2}g_i + r_{i,1}m_{i,2} + r_{i,2}m_{i,1})g_i + m_{i,1}m_{i,2}}{z^2},$$

so the size of the randomness in the product increases by $\rho+\alpha$ bits for each multiplication.

Recall from Section 2.1.2 that the noise component in a public key is bounded by $\ell 2^{2\rho+\alpha} + \tau n 2^{\rho+1} + n^2 2^{\mu+\rho+1}$. Since $\mu \geq \alpha + \rho + \lambda$, we have

$$\ell 2^{2\rho+\alpha} + \tau n 2^{\rho+1} + n^2 2^{\mu+\rho+1} \leq 4n^2 2^{\mu+\rho},$$

which is $2 + 2\log_2(n) + \mu + \rho$ bits. Then the noise in a product of κ public keys is at most $\kappa(2 + 2\log_2(n) + \mu + \rho + \alpha) - \alpha$ bits. Again, recall from Section 2.1.2 that the noise component in a secret key is bounded by $\ell 2^\rho$, so that multiplying a product of κ public keys by a secret key yields an encoding whose noise is bounded by

$$\kappa(2 + 2\log_2(n) + \mu + \rho + \alpha) - \alpha + \log_2(\ell) + \rho + \alpha + 1$$

$$= \kappa(2 + 2\log_2(n) + \mu + \rho + \alpha) + \log_2(\ell) + \rho + 1$$

bits.

Remark 2.3.1 (Selecting ρ_f for other applications)**.** In CLT Diffie-Hellman, the particular level-1 encodings that get multiplied to generate the shared secret have a complicated bound on their noise. This is because the level-1 encodings are generated by adding subset sums of other level-1 encodings. There are applications of the CLT GES where it is easier to generate level-1 encodings. In essentially any application that does not require re-randomization or public sampling, the noise in a level-1 encoding will be bounded simply by 2^ρ. Then a level-κ encoding that is a product of level-1 encodings will have noise ρ_f bounded by $(\kappa-1)(\rho+\alpha)+\rho = \kappa\rho + (\kappa-1)\alpha$ bits. Crucially, ρ_f depends linearly on κ, regardless of the application. This is a major obstacle to practical application of the CLT GES.

Security

α is the bit-length of the g_i's. CLT take $\alpha = \lambda$ so that the order of the group $\mathbb{Z}_{g_1} \times \cdots \times \mathbb{Z}_{g_n}$ has no small prime factors. This is ostensibly necessary to prove that the security of a multi-party Diffie-Hellman key exchange that arises from a Graded Encoding Scheme reduces to the intractability of the GDDH problem. There is currently no known attack that exploits small g_i's.

ρ is the bit-length of the randomness used in the published level-0 encodings, and published level-1 encodings of $\mathbf{0}$. ρ must be large enough to withstand the following gcd attack:

Given two level-1 encodings of $\mathbf{0}$

$$c_1 \equiv \frac{r_{i,1}g_i}{z} \bmod p_i \quad \text{and} \quad c_2 \equiv \frac{r_{i,2}g_i}{z} \bmod p_i,$$

compute

$$u = \frac{c_1}{c_2} \bmod x_0.$$

Then for all i,

$$u \equiv \frac{r_{i,1}}{r_{i,2}} \bmod p_i,$$

whence

$$\gcd(ur_{i,2} - r_{i,1}, x_0) = p_i.$$

Then, for all possible $r_{i,1}$ and $r_{i,2}$, compute $\gcd(ur_{i,2} - r_{i,1}, x_0)$. The complexity of this attack if the gcd is computed naively is $\mathcal{O}(2^{2\rho})$. This batch gcd computation can be done in time $\mathcal{O}(\rho^2 2^\rho)$ [22]. Lee and Seo propose a different attack with complexity $\mathcal{O}(2^{\frac{\rho}{2}})$ [42]. CLT claim that this attack has large overhead, so they take $\rho = \lambda$ [25].

β is the bit-length of the random h_i's that appear in the zero testing parameter. The same gcd attack works on the zero-testing parameter. Instead of computing $u = \frac{c_1}{c_2}$, compute $u = \frac{(p_{zt})_1}{(p_{zt})_2} \equiv \frac{h_{i,1}}{h_{i,2}} \bmod p_i$, and compute $\gcd(h_{i,2}u - h_{i,1}, x_0)$ for all possible $h_{i,1}$,$h_{i,2}$. To avoid this attack, set $\beta = \lambda$.

n is the number of prime factors in the modulus x_0. CLT calculate that a traditional orthogonal-lattice attack on a set of level-0 encodings of $\mathbf{0}$ takes time $2^{\Omega(\frac{n}{\eta})}$. Roughly, given a set of integers $\{x_j\}$, such that $x_j \equiv r_{ij}g_i \bmod p_i$, find a short lattice vector $\mathbf{u}$ orthogonal to $\mathbf{x} \mod x_0$. Then $\mathbf{ux} \equiv 0 \mod p_i$, so $\mathbf{ur_ig_i} \equiv 0 \bmod p_i$. Since $\mathbf{u}$ is short enough, this holds over $\mathbb{Z}$, and hence $\mathbf{r_ig_i}$ can be computed easily. Then $\gcd(x_j - r_{ij}g_i, x_0) = p_i$, and as established earlier, the attacker can recover z. Thus we must have $n = \Omega(\eta \cdot \lambda)$.

Remark 2.3.2 (Incorrect parameter setting for n)**.** CLT mistakenly use the parameter setting $n = \omega(\eta \log_2 \lambda)$ instead of $n = \Omega(\eta \cdot \lambda)$. Their analysis cites [44] which also makes this error. This incorrect parameter setting persists in CLT's most recent multilinear map construction [25] which was designed to avoid attacks from Cheon et al. [19]. Note that [19] uses the correct parameter setting for n.

Finally, this error is also present in Apon et al.'s implementation of obfuscation [27]. All of [22], [25], and [27] provide running times and storage sizes for their implementations. Due to the incorrect choice for n in these implementations, their efficiency estimates are somewhat optimistic.

Efficiency

The CLT paper is entitled "*Practical* Multilinear Maps over the Integers", and provides some timing data for their implementation of a CLT Diffie-Hellman key exchange. The reader must be careful not to misinterpret their meaning of "Practical". After the authors of [22] make a few optimizations, they are able to achieve a public key size of 2.6 Gigabytes and a setup time of 7.5 hours for a low security level and a small number of parties (while also choosing an insufficiently large value for n). See Example 2.3.4 for specifics.

A good way to understand the efficiency of the CLT scheme is to examine concrete parameter sets.

Example 2.3.3 (CLT parameter set)**.** For $\lambda = 128$ and $\kappa = 10$, we have the following selection of CLT parameters:

- $\alpha = \rho = \beta = 128$.
- $\rho_f = \kappa\lambda + (\kappa - 1)\lambda = 2432$. Note that this is using the smaller estimate of ρ_f based on level-1 encodings having noise bounded by 2^ρ (cf. Remark 2.3.1).

- $\eta = \rho_f + 4\lambda + 8 = 2952$.
- $n = \eta \cdot \lambda = 377,856$.
- x_0 will be an $n\eta \approx 1.1$ billion bit integer.
- $\ell = n\alpha + 2\lambda \approx 48.3$ million.
- $\tau = n(\rho + \log_2(n) + 1) + 2\lambda \approx 55.7$ million.

The TTP publishes ℓ level-0 encodings, $\tau + n$ level-1 encodings of $\mathbf{0}$, and one level-1 encoding of $\mathbf{1}$. These are all integers modulo x_0. They also publish p_{zt}, which is comprised of n integers modulo x_0. So in total, the TTP publishes about 104.7 million 1.1 billion bit integers, which is roughly 14.4 petabytes of data.

An individual user's public key is a single level-1 encoding, and is therefore about 137 megabytes.

Remark 2.3.4 (Bitlength of x_0 depends quadratically on κ and λ)**.** The biggest culprits of these large numbers are the quadratic dependency of $n\eta$ on κ, the cubic dependency on λ, and the large number of encodings that must be published in order to enable public sampling and re-randomization. The quadratic dependency of $n\eta$ on κ is a fundamental consequence of the design:

$$n\eta = \eta^2 \cdot \lambda = (\rho_f + 4\lambda + 8)^2 \cdot \lambda = ((2\kappa - 1)\lambda + 4\lambda + 8)^2 \cdot \lambda = \mathcal{O}(4\kappa^2\lambda^3).$$

It arises because in order for the zero-tester to function, no amount of multiplications can cause the numerator of the encodings to roll over any of the moduli p_i. To compensate for this, the p_i must be made large enough so that after the desired number of multiplications, the numerators of encodings will still be smaller than p_i. Basically, the noise grows with each multiplication, and eventually overwhelms the signal, so the noise must be very small compared to p_i initially. But the noise cannot be too small, because then there are easy attacks, as discussed above. The result is that η has a linear dependency on κ, and hence $n\eta$ has a quadratic dependency.

Some applications require very large multilinearity, and for those applications, the noise-growth problem is an overwhelming obstacle to practical implementation.

Heuristic Optimizations

CLT offer some heuristic optimizations to improve the size of the public parameters. The TTP can simply publish fewer encodings. The cost of doing so is that the proofs that the sampling procedure and the re-randomization procedure produce encodings of sufficiently random messages no longer hold. CLT set $\ell = 2\lambda$ instead of $\ell = n\alpha + 2\lambda$, and instead of storing $\tau = n(\rho + \log_2(n) + 1) + 2\lambda$ re-randomization encodings, they store $\sqrt{n}$ level-0 encodings, and $\sqrt{n}$ level-1 encodings of $\mathbf{1}$. These can be combined by users running the re-randomization algorithm to create n^2 re-randomization encodings of $\mathbf{0}$ at level-1.

The final adjustment they suggest is using a zero-testing integer rather than a zero-testing vector:

$$p_{zt} = \sum_{i=1}^{n} h_i (z^k g_i^{-1} \bmod p_i) \frac{x_0}{p_i} \mod x_0.$$

It is still the case that if c is a level-κ encoding of $\mathbf{0}$ then cp_{zt} is small. However, it is no longer the case that if cp_{zt} is small, and c is a level-κ encoding, then c is a level-κ encoding of $\mathbf{0}$. This means that in a CLT Diffie-Hellman key exchange, there are multiple messages whose level-κ encodings will extract to the shared secret. According to CLT, this is not a problem [22].

Making these changes, the public key is only $2\lambda + 2\sqrt{n} + 2$ integers modulo x_0, which with the parameter set in Example 2.3.3 is roughly 872 encodings rather than 104.7 million. 872 1.1 billion bit integers is about 121 gigabytes of data instead of 14.4 petabytes. Note that the alteration to the zero-testing parameter has a relatively small effect on the size of the public parameters compared to the alteration to the number of published encodings. However, having a zero-testing integer instead of a zero-testing vector means that users computing their shared secret only need to do one multiplication of two $n\eta$ bit numbers, instead of n multiplications. This is a fairly significant speed improvement considering how large n is, and how expensive it is to multiply $n\eta$-bit integers.

Example 2.3.5 (CLT proof of concept parameter set)**.** In [22], CLT implement their multilinear map and provide some timing data for their implementation. By using their heuristic optimizations, they were able to use the following parameter set:

- $\lambda = 80$
- $\kappa = 6$ (7 parties)
- $n = 26115$ (This choice of n is too small; see Remark 2.3.2)

- $\eta = 2438$.

According to [22], this parameter set resulted in a public key size of 2.6 Gigabytes, a setup time of 27295 seconds, a publish time of 17.8 seconds, and a key generation time of 20.2 seconds (16-core Intel Xeon CPU E7-8837 at 2.67GHz). However, recall that there is a lattice attack that runs in time $2^{\Omega(\frac{n}{\eta})}$, which for this parameter set is $2^{10.7} \ll 2^{80}$. The hidden constant in the attack running time must be larger than 2^{70} before this parameter set achieves 80-bit security.

2.3.3 Public Key Validation

The first external cryptanalysis of the CLT scheme was done by Lee and Seo [42]. Most of their paper focuses on improving the gcd attack mentioned in Section 2.3.2. However, the paper also contains an interesting attack on the CLT Diffie-Hellman protocol in the case where users fail to validate public keys they receive. It is an attack by a dishonest user who generates an invalid public key, and uses the resulting shared secret computation to learn bits of another user's private key. It applies in particular to the version of CLT that uses a zero-testing integer rather than a zero-testing vector.

Checking the validity of public keys is therefore necessary for security. However, so far, no one has provided a way to do so. This section describes the Lee-Seo attack, and describes a heuristic method for validating public keys in a CLT Diffie-Hellman protocol.

Malicious User Attack

An outline of the Lee-Seo attack follows.

Suppose there are N users, and the adversary wants to recover the private key of user 1. Let the public keys be c'_i and the private keys be c_i. Instead of publishing c'_N, the adversary publishes $c''_N = c'_N + 2^k X \mod x_0$ where k is some small integer less than ν, and X is a level-1 encoding of 0. Note that the shared secret sk that user 1 computes is the ν most significant bits of

$$\begin{aligned} &\left(c_1 c''_N \prod c'_j\right) \cdot p_{zt} \bmod x_0 \\ = &\left(c_1 c'_N \prod c'_j\right) \cdot p_{zt} + \left(c_1 2^k X \prod c'_j\right) \cdot p_{zt} \bmod x_0. \qquad (\star) \end{aligned}$$

The adversary can compute the ν most significant bits of the first term of ($\star$) as the ν most significant bits of

$$\left(c_1' c_N \prod c_j'\right) \cdot p_{zt} \bmod x_0.$$

The adversary's goal is to compute

$$\left(c_1 X \prod c_j'\right) \cdot p_{zt} \bmod x_0$$

so that they can divide by

$$\left(X \prod c_j'\right) \cdot p_{zt} \bmod x_0$$

to recover c_1. The ν most significant bits of $\left(c_1 X \prod c_j'\right) \cdot p_{zt} \bmod x_0$ are 0 since this is a level-N encoding of zero, which means that the ν most significant bits of the second term of ($\star$) has at most k non-zero trailing bits. Then there are only k bits of sk that remain unknown to the adversary.

Suppose that the adversary obtains some $(m, t = \mathrm{MAC}_{sk}(m))$ pairs from user 1. Here MAC is a symmetric-key message authentication scheme. The adversary can use these (m, t) pairs to check guesses for sk. Since k is small, the adversary will succeed and therefore learn the ν most significant bits of $\left(c_1 2^k X \prod c_j'\right) \cdot p_{zt} \mod x_0$, the k trailing bits of which are the k most significant bits of $\left(c_1 X \prod c_j'\right) \cdot p_{zt} \mod x_0$.

The get the next k bits, perform the same attack, but set the adversary's public key to $c_N' + 2^{2k} X \mod x_0$ (it will be necessary to collect new (m, t) pairs for checking guesses of the new sk). Repeating this method yields the first 2ν bits of $\left(c_1 X \prod c_j'\right) \cdot p_{zt} \mod x_0$. A slightly different method can be used to recover the remaining bits. See [42] for details.

A Heuristic Public Key Validation Technique

Recall that a valid public key is an integer $c \in \mathbb{Z}_{x_0}$ of the form

$$c \equiv \frac{r_i g_i + m_i}{z} \bmod p_i$$

for all i, where $m_i \in \mathbb{Z}_{g_i}$ and r_i is small relative to p_i.

Notice that for any $c \in \mathbb{Z}_{x_0}$ and for all i,

$$c = \frac{a_i}{z} \bmod p_i$$

for some $a_i \in \mathbb{Z}_{p_i}$. For any integer a_i, by the division algorithm there exist unique R_i and $m_i \in \mathbb{Z}_{g_i}$ such that $a_i = R_i g_i + m_i$. Since g_i is small relative to p_i, there are unique $R_i \in \mathbb{Z}_{p_i}$ and $m_i \in \mathbb{Z}_{g_i}$ such that

$$c \equiv \frac{R_i g_i + m_i}{z} \mod p_i$$

for all i. That is, every $c \in \mathbb{Z}_{x_0}$ can be written in the form of a level-1 encoding in exactly one way. Then the only way such an integer can fail to be a valid public key is for one of the resulting R_i's to be larger than the acceptable bound. This reduces the problem of public key validation to the problem of checking whether the R_i's corresponding to a given public key c are too large.

One technique that could be used is the following: Given an integer c, consider $bc^{\kappa-1}$ where b is a level-1 encoding of 0. If c is a valid level-1 encoding of some message, then $bc^{\kappa-1}$ is a valid level-κ encoding of 0, and hence returns "true" when zero-tested. Conversely, suppose that c is an invalid level-1 encoding. Then it is expected that $bc^{\kappa-1}p_{zt} \mod x_0$ will exceed $x_0 2^{-\nu}$ bits. Note that this is not guaranteed, due to the heuristic component of the "single-integer" zero-tester. To increase confidence in this test, one could perform it with multiple different random b's, and adjust the relative encoding levels of the c's and b's.

The only part of the invalid encoding an attacker controls in the Lee-Seo attack is the extra bit-length of the randomness (otherwise, the encodings are chosen randomly). But it is unlikely that this validation fails with high probability on a particular length of invalid encoding, especially since the validator can adjust the length by adjusting the relative levels of c and b.

2.3.4 Zero-izing Attacks

The next major cryptanalysis of CLT came from Cheon et al. [19] who describe an attack on CLT that exploits published level-1 encodings of $\mathbf{0}$ to recover all the secret parameters. This attack breaks the CLT scheme described in Section 2.1 completely for applications where level-1 encodings of $\mathbf{0}$ are published (Diffie-Hellman for example). Applications that do not need re-randomization (and hence do not publish level-1 encodings of $\mathbf{0}$) such as obfuscation and broadcast encryption (cf. Chapters 3 and 4, respectively) may not be affected. However, it is conceivable that level-1 encodings of $\mathbf{0}$ are attainable from the public information provided in these applications.

Furthermore, the Cheon et al. attack has since been extended by Gentry et al. [36] to work given many level-1 encodings whose product is an encoding of $\mathbf{0}$, instead of many level-1 encodings of exactly $\mathbf{0}$. Such a set of encodings is called *orthogonal*. In order to truly have confidence that applications not publishing low-level encodings of $\mathbf{0}$ remain resistant to the Cheon et al. attack and subsequent extensions, it would be necessary to prove that no set of orthogonal encodings is computable from the public information in the application. This will be discussed further in Chapters 3 and 4.

The remainder of this section describes the zero-izing attack of Cheon et al. See [36] for details on the extension to orthogonal encodings.

Preliminaries

The attack works by expressing the process of zero-testing several level-κ encodings of $\mathbf{0}$ in terms of matrix multiplication. Since for any level-κ encoding u of $\mathbf{0}$, up_{zt} is small, the resulting matrix equation will be satisfied over $\mathbb{Q}$ instead of just modulo x_0. This enables eigenvalue computation, and it happens that the eigenvalues of the matrix reveal the secret parameters.

Let c be any level-0 encoding, with $c \equiv c_i \bmod p_i$ for all $i \leq n$. Recall that $\{x'_j\}$ is a set of ℓ public level-0 encodings, $\{x_k\}$ is a set of τ public level-1 encodings of $\mathbf{0}$, and Y is a public level-1 encoding of $\mathbf{1}$. Write $Y \equiv \frac{y_i g_i + 1}{z} \bmod p_i$, $x'_j \equiv x'_{i,j} \bmod p_i$, and $x_k \equiv \frac{r_{i,k} g_i}{z} \bmod p_i$ for all i. Then for all $j \leq \ell$, $k \leq \tau$,

$$cx'_j x_k Y^{\kappa-1} \mod x_0$$

is a level-κ encoding of $\mathbf{0}$. Note that this can be computed entirely from public parameters. For a fixed c, the attacker can compute

$$\begin{aligned} \omega_{j,k} &= cx'_j x_k Y^{\kappa-1} (p_{zt})_1 \mod x_0 \\ &= cx'_j x_k Y^{\kappa-1} \sum_{i=1}^{n} h_{i,1} (z^\kappa g_i^{-1} \bmod p_i) \frac{x_0}{p_i} \mod x_0. \end{aligned}$$

Since for all i,

$$cx'_j x_k Y^{\kappa-1} = \frac{c_i x'_{i,j} r_{i,k} g_i (y_i g_i + 1)^{\kappa-1}}{z^k} + Q_i p_i$$

for some Q_i, we have

$$\begin{aligned}\omega_{j,k} &= \sum_{i=1}^{n} h_{i,1}\left(c_i x'_{i,j} r_{i,k} (y_i g_i + 1)^{\kappa-1} \bmod p_i\right) \frac{x_0}{p_i} + \sum_{i=1}^{n} Q'_i x_0 \mod x_0 \\ &= \sum_{i=1}^{n} h_{i,1} c_i x'_{i,j} r_{i,k} (y_i g_i + 1)^{\kappa-1} \frac{x_0}{p_i} \mod x_0 \\ &= \sum_{i=1}^{n} h'_i c_i x'_{i,j} r_{i,k} \mod x_0,\end{aligned}$$

where $h'_i = h_{i,1}(y_i g_i + 1)^{\kappa-1} \frac{x_0}{p_i}$.

Since $cx'_j x_k Y^{\kappa-1}$ is a level-κ encoding of $\mathbf{0}$, $cx'_j x_k Y^{\kappa-1}(p_{zt})_1$ is much smaller than x_0 (indeed, p_{zt} is chosen specifically so that this is the case). So the equation above is true over $\mathbb{Z}$ instead of just $\mathbb{Z}_{x_0}$. Then

$$\omega_{j,k} = \sum_{i=1}^{n} h'_i c_i x'_{i,j} r_{i,k}.$$

For a fixed c, and fixed n-subsets of $\{x'_j\}$ and $\{x_k\}$, J and K respectively, this equation can be expressed as matrix multiplication. Consider the $n \times n$ matrix $W_c^{J,K}$ whose $(j,k)^{th}$ entry is ω_{J_j,K_k}:

$$W_c^{J,K} = X'HCR,$$

where $X'_{i,j} = x'_{i,J_j}$, H is the diagonal matrix with entries h'_i, C is the diagonal matrix with entries c_i, and $R_{i,k} = r_{i,K_k}$. For a fixed c, an attacker can compute many such matrices by varying J and K. Note that H is constant for all choices of c, J, and K. Also, X' and R depend only on J and K, and not on c.

Attack Procedure

The attacker sets $c = x'_1$, and chooses J and K such that $W_c^{J,K}$ is invertible (this can be done in time $\mathcal{O}(\kappa^{\omega+3}\lambda^{2\omega+6})$; for details, see [19]). Then $W_d^{J,K}$ is also invertible for all d, since if $W_c^{J,K} = X'HCR$, then X' and R are invertible, so $W_d^{J,K} = X'HDR$ is also invertible.

The attacker sets $d = x'_2$, and computes

$$\begin{aligned}W_c^{J,K}(W_d^{J,K})^{-1} &= X'HCRR^{-1}D^{-1}H^{-1}(X')^{-1} \\ &= X'CD^{-1}(X')^{-1}.\end{aligned}$$

Then the eigenvalues of $W_c^{J,K}(W_d^{J,K})^{-1}$ are equal to the entries of CD^{-1}. The eigenvalues can be computed in polynomial time, revealing $\frac{x'_{i,1}}{x'_{i,2}}$ to the attacker. Write $\frac{x'_{i,1}}{x'_{i,2}}$ in reduced form as $\frac{x''_{i,1}}{x''_{i,2}}$. The attacker can easily recover $x''_{i,1}$ and $x''_{i,2}$ from their ratios. Now

$$\frac{x'_1}{x'_2} \equiv \frac{x''_{i,1}}{x''_{i,2}} \mod p_i,$$

so

$$x''_{i,2}x'_1 - x'_2x''_{i,1} \equiv 0 \mod p_i.$$

Since the attacker knows $x''_{i,1}$, $x''_{i,2}$, x'_1, and x'_2, she can compute

$$\gcd(x''_{i,2}x'_1 - x'_2x''_{i,1}, x_0)$$

to recover p_i for all i.

Recall from Section 2.1.1 that given the factorization of x_0, one can use rational reconstruction to compute the remaining secret parameters of the CLT GES.

2.3.5 Defending Against Zero-izing Attacks

The Cheon et al. attack is a complete break of the CLT GES as described in Section 2.2. Following the publication of this attack, there were two independent attempts to repair the CLT multilinear map by Garg, Gentry, Halevi and Zhandry [34], and Boneh, Wu, and Zimmerman [13]. In [24], Coron, Lepoint and Tibouchi demonstrate that these candidate fixes fail to prevent Cheon et al.'s attack. Similarly, Brakerski et al. show that repairs of the original GGH multilinear map are also ineffective [16]. In [25], Coron, Leopint and Tibouchi give their own fix of CLT that so far has resisted further zero-ization attempts. The fixed version of CLT is known as CLT15.

The new CLT multilinear map avoids zero-ization using a new modulus N for p_{zt}. So p_{zt} is reduced modulo N instead of modulo x_0. This ensures that for an encoding c, the components of $c \cdot p_{zt} \mod N$ depend nonlinearly on the CRT components of $c \mod x_0$. The zero-ization attack relies heavily on these two objects having a *linear* relationship, and so the attack does not apply to new CLT.

For the complete construction, see [25].

Chapter 3

Code Obfuscation

For expository purposes, Graded Encoding Schemes have up to this point been described in the context of multi-party key exchanges. There are a variety of other ways to perform a multi-party key exchange (though, none in only one round), most of which are vastly more practical than using multilinear encodings. The reason there is interest in GESs is not the multi-party key exchanges they induce, but the new applications they enable that have otherwise been unattainable. Chief among these applications is the subject of this chapter: general-purpose code obfuscation.

Roughly speaking, a *virtual black box* (VBB) obfuscator O is a program that takes as input a program P, and outputs a program $O(P)$ equivalent to P that "reveals nothing" about P except what could be learned with oracle access to P.

Such a thing would be very useful. For example, suppose for a given secret s, P_s is the program that returns 1 if given s as input and 0 otherwise (the function this program computes is known as a *point function*). A user Alice who knows s can easily generate P_s (she can simply hardcode a check that the input is equal to s). If for some reason Alice wanted Bob to be able to run P_s, she cannot necessarily publish P_s, as it may be possible to recover s from the implementation of P_s. However, if Alice has access to an obfuscator O, she can publish $O(P_s)$ because by definition, $O(P_s)$ provides no information about P besides what could be learned from having oracle access to P. Then in order to learn s from $O(P_s)$, one would have to guess s.

As a second example, suppose that for some reason, we do not trust traditional public key cryptography based on RSAP or ECDLP, and that we would really prefer to use symmetric key cryptography. Unfortunately for us, it seems difficult to achieve many of the nice key management properties of public key crypto with only existing symmetric encryption schemes. General program obfuscation gives a way to convert a symmetric key encryption scheme into a public key encryption scheme as follows.

Let E_s and D_s be the encryption and decryption algorithms respectively for a symmetric scheme S. Consider the public key scheme P where a user i's secret key is a random key k_i from the keyspace of S, and their public key is $P = O(E_s(k_i, \cdot))$. Here P is the obfuscation of the program that takes in a plaintext m, and outputs $E_s(k_i, m)$. To encrypt a message m for user i, evaluate user i's public key P on the input m to compute $c = P(m) = E_s(k_i, m)$. Then user i can decrypt in the obvious way, by running $D_s(k_i, c)$.

Since P is obfuscated, it reveals nothing about the circuit $E_s(k_i, m)$. In particular, an eavesdropper is unable to learn any information about k_i or m besides what it could learn from having access to an encryption oracle, which it has access to by the definition of a public key scheme.

Example 3.0.6 (Elementary VBB obfuscator)**.** There is a trivial exponential-size VBB obfuscator due to Bernstein, Hülsing, Lange and Niederhagen [6]: Given a pseudorandom generator (see Section 4.2.1) $\mathrm{PRG}\colon \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$, to obfuscate a boolean function $f\colon \{0,1\}^\lambda \to \{0,1\}$, store in a table $\mathrm{PRG}(x)$ for all x where $f(x) = 1$. Additionally, pad the table to size 2^λ with random entries from $\{0,1\}^{2\lambda}$. With high probability, none of the random entries have a preimage under PRG, and therefore the fake entries in the table do not correspond to any input $x \in \{0,1\}^\lambda$. The obfuscation of f is the sorted table which has size $\lambda 2^{\lambda+1}$ bits. To evaluate $f(x)$, simply compute $\mathrm{PRG}(x)$, and check if it is in the table. If it is, $f(x) = 1$, otherwise, $f(x) = 0$.

Until the advent of cryptographic multilinear maps, no *polynomial* size general-purpose obfuscator was known to exist. It had been possible before to obfuscate small classes of programs (for example, it is easy to obfuscate point functions using cryptographic hash functions), but there was no way to obfuscate an arbitrary function. In fact, it had been proven that a general-purpose code obfuscator was impossible due to the existence of so-called "un-obfuscatable" functions [3].

In 2013, Garg, Gentry, Halevi, Raykova, Sahai and Waters published a candidate general-purpose obfuscator [32]. Their construction avoids the impossibility result by considering a slightly weaker notion of obfuscation called *indistinguishability obfuscation* (*iO*). This chapter contains a description of the Garg et al. candidate, some background information about the tools used in this construction, and some information on subsequent improvements in obfuscation constructions.

Note that while the Garg et al. construction is polynomial time and size, the exponential-size Bernstein et al. obfuscator is much more efficient in practice. Using the trivial method, obfuscating a function and storing the obfuscation take exponential time and space, but evaluating the obfuscated function requires only a table lookup. Furthermore, due to large constants in the Garg et al. construction, the trivial method can produce smaller obfuscations in many practical situations (cf. Example 3.3.2).

3.1 Background

The Garg et al. iO candidate employs the following general strategy:

It first converts P from a *circuit* into an equivalent matrix multiplication problem called a *branching program*. The goal of obfuscation is to ensure that the only thing that can be computed with knowledge of these matrices is the complete product, which could be computed with oracle access to P. Since multilinear encodings are in fact multilinear, encoding all the entries of the matrices produces a set of matrices whose product is an encoded version of the original solution. Encoding the matrices not only obscures their structure, but ensures that the intermediate matrices can only be meaningfully combined in the intended way.

This section discusses circuits and branching programs which are essential to the Garg et al. construction. This section also contains a discussion of the difference between iO and VBB.

3.1.1 Circuits

In order to create an obfuscator that works for any program, a general framework for describing programs is necessary. The Garg et al. obfuscation construction is based on obfuscating programs expressed as circuits:

Definition 3.1.1 (Boolean Circuit)**.** Let B be a basis of boolean functions. A *boolean circuit* with n inputs and m outputs is a finite, directed, acyclic graph such that:

i vertices of in-degree 0 are labeled as “input”, or are labeled by a constant boolean value;

ii vertices of out-degree 0 are labeled as “outputs”;

iii every other vertex is labeled by a function from B such that the in-degree of each vertex is equal to the number of arguments its label takes; and iv the number of vertices labeled as “input” and “output” is n and m respectively.

The vertices of a boolean circuit are referred to as *gates*. The *size* of a circuit is the number of gates in the circuit, and the *depth* of a circuit is the length of the longest path from an input vertex to an output vertex. The *fan-in* of a circuit is the maximum in-degree of any vertex.

A circuit is *evaluated* by assigning input values to the input nodes, and assigning to each out-neighbour of a vertex the result of the labeled function evaluated on the values of the in-neighbours. The output of a circuit evaluation is the values assigned to the nodes labeled “output”.

For fixed integers n and m, and any function $f : \{0,1\}^n \to \{0,1\}^m$, there is a boolean circuit whose evaluation on any input x is equal to $f(x)$.

Circuit Complexity

Algorithm complexity is often studied with respect to the asymptotic behaviour of the algorithm relative to the input size of the problem. A circuit, however, has a fixed input size. Circuit complexity is therefore studied in terms of *families* of circuits $\{C_\lambda\}$. A circuit family solves a particular problem (describes a “formal language”). Each C_λ solves instances of the problem where the input size is λ.

A circuit *class* $\{\mathcal{C}_\lambda\}$ is a set of circuit families. Each $\mathcal{C}_\lambda$ is a set of circuits that solve a problem in the class for input size λ.

The goal of general-purpose obfuscation is to be able to obfuscate the circuit class P: the set of all boolean circuit families $\{C_\lambda\}$ with fan-in 2 and such that the size of C_λ is a polynomial function of λ (indeed, every language in P is recognized by a polynomial sized circuit with fan-in 2).

Garg et al. achieve obfuscation for all such circuits by first creating an obfuscation algorithm for a smaller class of circuits, and then leveraging this algorithm along with fully homomorphic encryption to get an obfuscation algorithm for all circuits. The small class of circuits is called NC^1.

NC^1 is, roughly speaking, the set of circuit families with fan-in 2 and a single output such that the depth of $C \in \mathcal{C}_\lambda$ is $\mathcal{O}(\log \lambda)$, and the size of C is a polynomial function of λ. In other words, NC^1 is the class of decision problems solvable in parallel logarithmic time by a polynomial number of processors. Obfuscating all circuits in NC^1 is all that is necessary to extend obfuscation to P, including circuits with multiple output bits.

Universal Circuits

The task of obfuscating a circuit can be reduced to the task of obfuscating the input to a different circuit by using what are known as universal circuits.

Definition 3.1.2 (Universal Circuit)**.** Let $\{\mathcal{C}_\lambda\}$ be a circuit class. A family of circuits $\{U_\lambda\}$ where each U_λ takes as input a circuit description α for a circuit $C \in \mathcal{C}_\lambda$ and an input m is called a universal circuit family for $\{\mathcal{C}_\lambda\}$ if for all U_λ, C and m, $U_\lambda(\alpha, m) = C(m)$.

Intuitively it is easier to ensure that no bits of (α, m) are revealed than it is to ensure that no information about C is revealed. The universal circuit U_λ is simply a way to evaluate C given only α.

3.1.2 Branching Programs

Branching programs are another model of computation. They are more convenient than circuits, because evaluating a branching program amounts to performing a series of matrix multiplications, whereas circuit evaluation is more difficult to express as a multilinear operation.

Definition 3.1.3. Let A_0 and A_1 be two distinct permutation matrices. An *oblivious matrix branching program* of length n and input size m is a sequence

$$BP = ((inp(i), A_{i,0}, A_{i,1}))_{i=1}^{n}$$

where the $A_{i,b}$ are permutation matrices, and $inp : [n] \to [m]$ is the input-bit selection function. On an m-bit string input x, a matrix branching program evaluates to 0 if $\prod_{i=1}^{n} A_{i,x_{inp(i)}} = A_0$, 1 if this product is A_1, and is undefined otherwise.

Such a branching program is called *oblivious* because the function *inp* does not depend on the input x.

At each step i, the evaluator examines $inp(i)$, and chooses one of two fixed matrices $A_{i,0}$ or $A_{i,1}$ for that step based on the value of the input bit $x_{inp(i)}$. Then, the product of all chosen matrices is computed, and the evaluator returns 1 or 0 depending on the product. Note that it is possible for a step i to examine the same input bit as a different step j, so that the same bit of x can be examined multiple times during the evaluation of the branching program.

Note that if one were to encode all the entries of the matrices in a branching program with a GES and otherwise evaluate it normally, the result would be an encoding of A_0, or an encoding of A_1, by the multilinearity of matrix multiplication and GES. Multilinear maps do not interfere with the structure of a branching program, and it is this fact that enables obfuscation of NC^1 circuits.

The *width* of a branching program is the size of the $A_{i,j}$'s. Evaluating a length n branching program takes $n-1$ matrix multiplications. In order for the evaluation of a branching program to be feasible, the length and the width must be polynomial in the input size. We will see shortly that this is the reason for restricting to NC^1 circuits when designing obfuscation.

Barrington's Theorem

In order to obfuscate circuits, it is useful to convert them to branching programs. As it happens, it is possible to efficiently convert NC^1 circuits into polynomial length branching programs, due to a beautiful theorem of Barrington.

Theorem 3.1.4 (Barrington's Theorem [4])**.** For any circuit C of depth d with one output bit, there exists an oblivious matrix branching program B of width 5 and length at most 4^d such that B and C compute the same function.

Hence, all circuits of depth $\mathcal{O}(\log \lambda)$ can be expressed as matrix branching programs of polynomial length and fixed width. Furthermore, the proof of Barrington's Theorem is constructive, and gives an efficient algorithm for finding B given C.

Example 3.1.5 (Converting a NOT gate to a branching program)**.** Suppose that

$$M = (inp, A_0 = I, A_1, \{A_{i,b} \mid i \leq n, b \in \{0,1\}\})$$

is a length-n branching program computing the circuit C. Construct a length-n branching program M' that computes the circuit $\neg C$ by taking $A'_{inp(1),0} = A_1^{-1} \cdot A_{inp(1),0}$, $A'_{inp(1),1} = A_1^{-1} \cdot A_{inp(1),1}$, and $A'_{i,b} = A_{i,b}$ for all $i \neq inp(1)$.

If

$$M(x) = \prod_{i=1}^{n} A_{inp(i),x_{inp(i)}},$$

then

$$M'(x) = \prod_{i=1}^{n} A'_{inp(i),x_{inp(i)}} = A_1^{-1} \cdot \prod_{i=1}^{n} A_{inp(i),x_{inp(i)}}.$$

So if $M(x) = I$ then $M'(x) = A_1^{-1}$; and if $M(x) = A_1$ then $M'(x) = I$. Hence

$$M' = \left(inp, I, A_1^{-1}, \{A'_{i,b}\}\right)$$

computes $\neg C$.

Example 3.1.6 (Converting an AND gate to a branching program)**.** Let C and D be circuits. Construct a branching program for the circuit $C \wedge D$ as follows:

Let A_C and A_D be 5-cycles (in S_5) whose commutator $A_C \cdot A_D \cdot A_C^{-1} \cdot A_D^{-1}$ is also a 5-cycle.

For all branching programs in this example, assume that $A_0 = I$. Let M_C be a branching program computing C with $A_1 = A_C$, and let M_C^{-1} be a branching program computing C with $A_1 = A_C^{-1}$. Similarly, let M_D and M_D^{-1} be branching programs computing D with $A_1 = A_D$ and $A_1 = A_D^{-1}$ respectively.

Finally, let M be the branching program $M_C M_D M_C^{-1} M_D^{-1}$ (where branching programs are appended to one another in the obvious way). If $C(x) = 0$, then $M_C(x) = I$, so $M(x) = I \cdot M_D(x) \cdot I \cdot M_D^{-1}(x)$. But $M_D(x) \cdot M_D^{-1}(x) = I$ since if $D(x) = 0$, they're both I, and if $D(x) = 1$, $M_D(x) = A_D$, and $M_D^{-1}(x) = A_D^{-1}$. So $M(x) = I$. Similarly, if $D(x) = 0$, $M(x) = I$.

When $C(x) = D(x) = 1$, $M(x) = A_C \cdot A_D \cdot A_C^{-1} \cdot A_D^{-1}$, which by assumption is a 5-cycle, and hence a permutation matrix in S_5. So M is a branching program computing $C \wedge D$ with $A_0 = I$ and $A_1 = A_C \cdot A_D \cdot A_C^{-1} \cdot A_D^{-1}$.

Since any circuit can be expressed in terms of NAND gates, Examples 3.1.4 and 3.1.5 can be used recursively to generate a branching program for the whole circuit. Each AND gate conversion spawns four new branching programs, which is where the exponential increase in depth comes from. It is this exponential factor that restricts obfuscation using this technique to NC^1. A circuit with linear depth would have a corresponding branching program with exponential length, and hence require an exponential number of matrix products to evaluate. The resulting obfuscation scheme, which relies on computing this matrix product, could not then be considered efficient.

3.1.3 VBB and iO

In light of the unobfuscatable functions presented by Barak et al. in [3], one might wonder how Garg et al. can claim to have a general purpose obfuscator. The answer is that they use a slightly weaker notion of obfuscation. Whereas Barak et al. disprove the existence of a *virtual black box* obfuscator, Garg et al. provide a construction for an *indistinguishability* obfuscator.

Definition 3.1.7. A *Virtual-Black-Box (*VBB*) Obfuscator* is a probabilistic polynomial-time algorithm O such that:

i For every circuit C, $O(C)$ describes a circuit that computes the same function as C.

ii The size and running time of $O(C)$ are at most polynomially larger than C.

iii For any PPT A, there exists a PPT simulator S such that for all C,

$$|Pr(A(O(C)) = 1) - Pr(S^C(1^{|C|}) = 1)|$$

is negligible.

A program obfuscated with a virtual black box obfuscator reveals no more information about the obfuscated program than can be learned from having oracle access to the function that the program computes. The obfuscated program is essentially a black box representation of the underlying function. It turns out this is impossible to achieve because there exist functions $\{f_\lambda\}$ for which there is a predicate easily computable from any circuit that computes f_λ, but not from oracle access to f_λ [3]. That is, there are functions whose implementations necessarily leak information that would be unobtainable merely from having oracle access to the function.

On the other hand, a program obfuscated with an indistinguishability obfuscator should merely be indistinguishable from any other equivalent program that is also obfuscated with the indistinguishability obfuscator (provided the two equivalent programs are equal in size). That is, given two programs that compute the same function and given their obfuscations, it should be infeasible to tell which obfuscated program corresponds to which unobfuscated program.

Definition 3.1.8. An *Indistinguishability Obfuscator* (iO) is a PPT algorithm such that

i For all circuits C and all inputs x,

$$iO(C)(x) = C(x).$$

ii The size and running time of $iO(C)$ are at most polynomially larger than C.

iii For any PPT distinguisher D, for all pairs of circuits C_0, C_1 with $C_0(x) = C_1(x)$ for all x and $|C_0| = |C_1|$,

$$|Pr(D(iO(C_0)) = 1) - Pr(D(iO(C_1)) = 1)|$$

is negligible.

An "unobfuscatable" function f can be still be obfuscated by an indistinguishability obfuscator. This is because the aforementioned predicate easily computable from any circuit that computes f does not allow for distinguishing between different circuits that compute f (since the predicate is the same for all such circuits).

iO is strictly weaker than VBB because it is possible for $iO(P)$ to leak information about P as long as $iO(P')$ leaks the same information for all P' equivalent to P. Otherwise, one could distinguish between the two obfuscations based on what information was leaked. Since it is possible for $iO(P)$ to leak information about P, it is not obvious that iO is even useful. We will see in Chapter 4 that the best way to make use of iO is to prove there exists a circuit $iO(P')$ indistinguishable from $iO(P)$ that cannot possibly reveal particular information (usually because P' does not actually contain the information in question).

Finally, it is important to note that if there is an algorithm Q that is the "best" obfuscator (hides the most information about its input circuit) for a circuit C, then $iO(C')$ is indistinguishable from $iO(Q(C))$, where C' is the circuit C padded to the length of $Q(C)$. In this sense, iO is the "best possible" notion of obfuscation.

3.1.4 Fully Homomorphic Encryption

Definition 3.1.9 (Fully Homomorphic Encryption)**.** A *fully homomorphic encryption scheme* (FHE) is a set of polynomial time algorithms (**Encrypt**, **Decrypt**, **KeyGen**, **Evaluate**).

- **KeyGen** takes as input the security parameter λ, and outputs a secret key sk and a public key pk.
- **Encrypt** takes as input pk and a plaintext m, and outputs a ciphertext c.
- **Decrypt** takes as input sk and a ciphertext c, and outputs the corresponding plaintext m.
- **Evaluate** takes as input pk, a circuit C, and a set of ciphertexts $\{c_i\}$ and outputs an encryption of $C(c_1, \ldots, c_n)$.

Furthermore, the algorithms satisfy the following properties:

- Correctness: $\textbf{Decrypt}(sk, \textbf{Encrypt}(pk, m)) = m$.
- Correctness of Evaluation: For all circuits C, If $c_i = \textbf{Encrypt}(pk, m_i)$, then $\textbf{Decrypt}(sk, \textbf{Evaluate}(pk, C(c_1, \ldots, c_n))) = C(m_1, \ldots, m_n)$.
- Compactness: **Decrypt** can be expressed as a circuit of size polynomial in λ.

Obfuscation requires an IND-CPA secure FHE scheme. The IND-CPA security definition in the FHE setting differs slightly from the traditional public key definition. Here, the adversary is allowed not only to encrypt a polynomial number of messages, but also to run **Evaluate** polynomially many times on any set of polynomial-time circuits with polynomially many inputs.

The Garg et al. obfuscation algorithm uses fully homomorphic encryption to extend indistinguishability obfuscation from circuits in NC^1 to circuits in P. Crucially, they use the fact that fully homomorphic encryption schemes exist with a special form of the compactness property, namely that their decryption algorithm can be expressed as a circuit in NC^1. For example, the FHE schemes described in [17] have log-depth decryption circuits (see Section 5.2.1 of [17]).

3.2 The Garg et al. Indistinguishability Obfuscator

Garg et al. construct an obfuscation algorithm iO for all circuits in P by first constructing an obfuscation algorithm iO_{NC^1} for all circuits in NC^1, and then using iO_{NC^1} as a subroutine for iO. This section provides an overview of both these components.

3.2.1 Obfuscation for NC^1

Obfuscating an NC^1 circuit begins by using Barrington's Theorem to convert the circuit into an equivalent *oblivious matrix branching program* (see Section 3.1.2). The goal of obfuscation is to allow third party evaluation of the branching program while concealing all the matrices and preventing all computations on these matrices that do not correspond to a valid evaluation of the branching program. The two most important tools used to achieve this goal are GESs and branching program randomization.

Graded Encoding Schemes for Obfuscation

Let $M = (A_0, A_1, \{A_{i,b}\}, inp)$ be a length-n oblivious matrix branching program. Use an Asymmetric GES to encode the entries of $A_{i,b}$ at level e_i (recall that e_i is the i^{th} characteristic vector). Call the encoded matrices $A'_{i,b}$. Similarly, encode the entries of A_0 and A_1 at level-$\mathbf{1}$ to get A'_0 and A'_1. By the multilinearity of matrix multiplication and Asymmetric GESs, $\prod_{i=1}^{n} A'_{i,b_i}$ is a matrix whose entries are level-$\mathbf{1}$ encodings of the entries of $\prod_{i=1}^{n} A_{i,b_i}$. Then any user with access to a level-$\mathbf{1}$ zero-testing parameter can check equality between A'_0 and $\prod_{i=1}^{n} A'_{i,b_i}$ by zero-testing the matrix $\prod_{i=1}^{n} A'_{i,b_i} - A'_0$.

The use of a GES is thought to ensure that computing a non-multilinear function of the matrices results in meaningless output. Intuitively, while the Graded Encoding Scheme preserves the multilinear structure of the matrices, and thus preserves the functionality of the branching program, it is not expected to preserve any other behaviour. Computing a non-multilinear function of encoded matrices therefore should not help an attacker to compute a non-multilinear function of the plaintext matrices. The obfuscation literature generally assumes that the underlying GES does a good job of preserving exactly the multilinear structure of its inputs and nothing else.

The other benefit of an Asymmetric Graded Encoding Scheme is the leveled structure that ensures the only encodings that can be zero-tested are those at the right level. Given $\{A'_{i,b}\}_{i=1}^{n}$, the only way to produce a zero-testable product is by taking a product that considers each matrix exactly once: $\prod_{i=1}^{n} A'_{i,b_i}$. If say, A'_{n,b_n} is excluded from the product, the result will be an encoding at level $(1, 1, \ldots, 1, 0)$, which is not zero-testable. Similarly, if A'_{n,b_n} is included twice, the result will be an encoding at level $(1, 1, 1, \ldots, 2)$ which is also not zero-testable. So not only does a GES make non-multilinear attacks hard, but it also restricts the type of multilinear attacks that are possible.

Remark 3.2.1 (GES fails to conceal matrices)**.** While use of a GES appears to make non-multilinear attacks difficult, and a large class of multilinear attacks infeasible, it does not achieve all the goals of obfuscation. Crucially, a GES does a bad job of actually concealing the matrices. Consider the setting where an attacker wishes to discover the $A_{i,b}$'s given only the $A'_{i,b}$s, A'_0, A'_1, and all the relevant parameters needed to evaluate the encoded branching program. Since the $A_{i,b}$'s are permutation matrices, an adversary can employ the following attack:

Choose any fixed n-dimensional $\{0, 1\}$ vector $\mathbf{b}$. For each $i < n$, compute the sum s_i of the entries of the first column of A'_{i,b_i}. This sum is a level-e_i encoding of 1 since A_{i,b_i} is a permutation matrix. Then $S = \prod_{i=1}^{n-1} s_i$ is a level-$(1, 1, \ldots, 1, 0)$ encoding of 1. The attacker can then compute SA'_{n,b_n}, which is a matrix of level-$\mathbf{1}$ encodings of the entries of

A_{n,b_n}. Zero-testing the entries of SA'_{n,b_n} therefore completely reveals A_{n,b_n}, since A_{n,b_n} is a $(0,1)$-matrix.

Remark 3.2.2 (GES fails to enforce order)**.** A GES prevents the computation of products that do not include exactly one of the $A'_{i,b}$'s for each i, but does not prevent an attacker from computing the product out of order. Indeed, $\prod_{i=1}^{n} A'_{i,b_i}$ and $\prod_{i=n}^{1} A'_{i,b_i}$ are both matrices of level-$\mathbf{1}$ encodings, and hence zero-testable, but only the former is a legitimate product in the obfuscation setting.

Kilian's Randomization

In light of Remarks 3.2.1 and 3.2.2, a supplemental obfuscation technique is necessary. Obfuscation algorithms employ what's known as Kilian's randomization.

In 1988, Kilian constructed an *oblivious circuit evaluation* protocol based on oblivious transfer [39]. Roughly, the goal of oblivious circuit evaluation is for two parties, Alice and Bob, to evaluate a circuit on a joint input without Alice or Bob revealing their input to the other.

Oblivious circuit evaluation is similar to obfuscation. Suppose Alice is the "obfuscator", and Bob is the "evaluator". Let the circuit they are trying to evaluate be a *universal circuit* U. U takes as input C and m, where C is a description of the circuit Alice wishes to obfuscate, and m is the input on which Bob would like to evaluate C. U outputs $C(m)$. An oblivious circuit evaluation protocol where Alice's "input" is C, and Bob's "input" is m would allow Bob to evaluate $U(C,m) = C(m)$ without learning Alice's input C. In fact oblivious circuit evaluation can be relaxed for the purposes of obfuscation, because in the obfuscation setting it is unimportant to prevent Alice from learning Bob's input.

Kilian's oblivious circuit evaluation protocol first converts U into an equivalent width-5 length-n matrix branching program $M = (A_0, A_1, \{A_{i,b}\}, inp)$. For each $1 < i < n$, let R_i be a random permutation in S_5. Let $R_0 = R_n = I$. Then the *randomized* matrix branching program is

$$M_R = \{A_0, A_1, \{\tilde{A}_{i,b} = R_{i-1} A_{i,b} R_i^{-1}\}_{i=1}^{n}, inp\}.$$

Note that

$$\prod_{i=1}^{n} \tilde{A}_{i,b_i} = A_{1,b_1} R_1^{-1} \cdot R_1 A_{2,b_2} R_2^{-1} \cdot \cdots \cdot R_{n-1} A_{n,b_n} = \prod_{i=1}^{n} A_{i,b_i}$$

so that M_R computes the same function as M.

In Kilian's protocol, the randomized matrices corresponding to Alice's input (that is, matrices $\{\tilde{A}_{k,C_{inp(k)}}\}$ where $inp(k)$ corresponds to a bit from C rather than m) are fixed and given to Bob. Note that if Bob is given $\tilde{A}_{i,0}$, he is not given $\tilde{A}_{i,1}$, and that since the matrices have been randomized, Bob cannot tell which bit a given matrix corresponds to even with access to the $A_{i,b}$'s. Indeed, for a fixed input (C, m), Kilian shows how to simulate the randomized matrices knowing only the product, and not the plaintext matrices. Kilian's randomization therefore does a much better job of concealing the plaintext matrices than a GES does.

Furthermore, Kilian's randomization enforces the correct order on the computation of all multilinear forms. For example, computing $\prod_{i=n}^{1} \tilde{A}_{i,b_i}$ does not allow the attacker to also compute $\prod_{i=n}^{1} A_{i,b_i}$ because multiplying the $\tilde{A}_{i,b_i}$'s out of order will fail to cancel the R_i factors.

To summarize, Kilian's randomization in combination with an Asymmetric GES conceals the matrices of the branching program, makes attacks that do not involve computing multilinear forms difficult, and prevents an attacker from computing any multilinear form that does not respect the order or multilinear structure of the branching program.

The resulting obfuscation procedure is as follows:

Given a circuit C in NC^1 to obfuscate, find a universal circuit U for circuits of size $|C|$ that takes as input a circuit description C' of length ℓ and a circuit input m of length ℓ' and outputs the evaluation $C'(m)$ of the given circuit on the requested input. Suppose for ease of description that the first ℓ bits of the input to U correspond to C' and the next ℓ' bits of the input to U correspond to m. Use Barrington's Theorem to convert U into an equivalent oblivious matrix branching program of width 5 and length n: $M = (A_0 = I, A_1, \{A_{i,b}\}, inp)$.

Choose a large prime q that is smaller than the order of the message space of the GES (for CLT, $q < \prod g_i$). For each $1 < i < n$ choose a random 5 by 5 invertible matrix R_i with entries in $\mathbb{Z}_q$ and compute their inverses (note that unlike in Killian's randomization, the R_i's are not restricted to being permutation matrices). The randomized matrix branching program is $M_R = \{A_0, A_1, \{\tilde{A}_{i,b} = R_{i-1} A_{i,b} R_i^{-1}\}_{i=1}^{n}, inp\}$.

Next, use a GES to encode the entries of each $\tilde{A}_{i,b}$ at level e_i to get $A'_{i,b} = Encode(\tilde{A}_{i,b}, e_i)$. If C' is the circuit description of C that can be input to U, then the obfuscation of C is comprised of:

- $\{A'_{i,C'_{inp(i)}} \mid inp(i) \leq \ell\}$, the fixed set of matrices that are selected by the bits of C',
- $\{A'_{i,b} \mid inp(i) > \ell \text{ and } b \in \{0, 1\}\}$, the set of matrices that are operated on by the bits of m,
- a level-$\mathbf{1}$ encoding of A_1,

- all public parameters of the GES (including a zero-testing parameter at level-$\mathbf{1}$).

Note that only one of the matrices in each step of the branching program that corresponds to an input bit from C' is published, whereas both matrices are published for steps that correspond to an input bit from m.

To evaluate the obfuscated branching program on an input m, for each step i, choose the matrix $A'_{i,C'_{inp(i)}}$ if $inp(i) \leq \ell$ and choose the matrix $A'_{i,m_{inp(i)-\ell}}$ if $inp(i) > \ell$. Compute the product P of the chosen matrices in order from $i = 1$ to n. Finally, zero-test the matrix $P - A_1$. If $P - A_1$ is the all-zeros matrix as confirmed by the zero-test, then $C(m) = 1$. Otherwise, $C(m) = 0$.

Partial Evaluation and Mixed Input attacks

There are still two classes of forbidden multilinear forms an attacker can compute from a branching program obfuscated in the manner above.

First, consider a randomized branching program

$$M_R = \{A_0, A_1, \{\tilde{A}_{i,b} = R_{i-1}A_{i,b}R_i^{-1}\}_{i=1}^{n}, inp\}.$$

An attacker can choose two different inputs (y, y') to this branching program, and evaluate the first j steps of the branching program on each input. This yields $P_y R_j^{-1}$ and $P_{y'} R_j^{-1}$. Since the random mask is the same for both inputs, the attacker can learn whether or not the intermediate results of the branching program on y and y' are the same by simply comparing $P_y R_j^{-1}$ to $P_{y'} R_j^{-1}$. This is called a *partial evaluation* attack.

Remark 3.2.3 (Partial evaluation attacks on encoded branching programs)**.** Discussion of partial evaluation attacks in [31] is limited to partial evaluation attacks against branching programs that have been randomized, but not encoded. A partial evaluation attack is more difficult to mount against a branching program that has been randomized and encoded than one that has only been randomized. This is because checking equality of two encoded matrices requires zero-testing. Checking equality of two partial evaluations P_1 and P_2 amounts to zero-testing $P_1 - P_2$. However, P_1 and P_2 are encoded below the zero-testing parameter (otherwise they would be complete evaluations), rendering them impossible to directly zero-test.

Instead the encoded matrix $P_1 - P_2$ must have its level raised to the zero-testing level via multiplication by an appropriate level encoding. Concretely, if P_1 and P_2 are level-$\sum_{i=1}^{j} e_i$ encodings, we can check if $P_1 = P_2$ by finding a level-$\sum_{i=j+1}^{n} e_i$ encoding c of a non-zero element $m \in \mathbb{Z}_q$, and zero-testing $(P_1 - P_2) \cdot c$ which is a matrix whose entries are level-**1** encodings. To generate c, choose a random entry from each $A'_{i,0}$ for $j < i \leq n$, and compute their product. Since the $A'_{i,0}$'s are encodings of uniformly random matrices, m is non-zero with high probability.

An attacker can also compute an illegal multilinear form by disregarding input consistency as she evaluates the branching program. At each step, the branching program examines a particular bit of the input. Often, two different steps will examine the same input bit. Suppose step i and j both examine the same input bit (that is, $inp(i) = inp(j)$), but the attacker chooses the matrices $A'_{i,0}$, and $A'_{j,1}$ instead of $A'_{i,0}$ and $A'_{j,0}$ or $A'_{i,1}$ and $A'_{j,1}$. Then the resulting product will compute a multilinear form encoded at level **1** that does not correspond to any honest evaluation of the branching program. This attack therefore reveals some of the branching program's non-oracle behaviour. This is known as a *mixed input* attack.

Preventing Partial Evaluation

Garg et al.'s strategy for defeating partial evaluation is to introduce randomization that is only cancelled on the ends of the branching program, in contrast to Kilian's randomization, which is cancelled by intermediate computations. Specifically, each 5 by 5 matrix $A_{i,b}$ of the original branching program is replaced by a $2m+5$ by $2m+5$ matrix $D_{i,b}$ for some m:

$$D_{i,b} = \begin{bmatrix} d_{i,b} & 0 \\ 0 & A_{i,b} \end{bmatrix}$$

where each $d_{i,b}$ is a random $2m$ by $2m$ diagonal matrix with entries in $\mathbb{Z}_q$. Garg et al. precautionarily take $m = 2n+5$, though they mention that they are unaware of any attack that arises if $m = 1$.

Then $\prod_{i=1}^{n} D'_{i,m_{inp(i)}}$ is an encoding of

$$R_0 \cdot \begin{bmatrix} d_m & 0 \\ 0 & \prod_{i=1}^{n} A_{i,m_{inp(i)}} \end{bmatrix} \cdot R_n^{-1}.$$

Before, R_0 and R_n were chosen to be the identity. Here, choose them randomly as with the other R_i's. In order to allow users to cancel R_0 and R_n, choose two random 5-vectors $\mathbf{s}$ and $\mathbf{t}$. Include in the obfuscation encodings $\hat{\mathbf{s}}$, $\hat{\mathbf{t}}$ of two "bookend" $(2m+5)$-vectors $\mathbf{s}^* = (\mathbf{0}, \mathbf{r_s}, \mathbf{s}) \cdot R_0^{-1}$ and $\mathbf{t}^* = R_n \cdot (\mathbf{r_t}, \mathbf{0}, \mathbf{t})^T$. Here, $\mathbf{r_s}$ and $\mathbf{r_t}$ are random m-vectors. Let

$\mathbf{s}^*$ and $\mathbf{t}^*$ be encoded at levels e_1 and e_{n+2} respectively (and consequently, encode $\tilde{D}_{i,b}$ at level e_{i+1} instead of e_i). Finally, include a level-$\mathbf{1}$ encoding of $p' = \langle \mathbf{s}, \mathbf{t} \rangle$.

To evaluate the obfuscated circuit on the input m, compute $\hat{\mathbf{s}} \cdot \prod_{i=1}^{n} D'_{i,m_{inp(i)}} \cdot \hat{\mathbf{t}}$ which is an encoding of

$$\begin{bmatrix} 0 & 0 \\ 0 & \mathbf{s} \cdot \prod_{i=1}^{n} A_{i,m_{inp(i)}} \cdot \mathbf{t} \end{bmatrix}.$$

If $A_m = \prod_{i=1}^{n} A_{i,m_{inp(i)}}$ is the identity, then $\mathbf{s} \cdot A_m \cdot \mathbf{t} = \langle \mathbf{s}, \mathbf{t} \rangle$, which can be compared to p' via the zero-testing parameter.

Remark 3.2.4 (Intuition for the partial evaluation defense)**.** Suppose an attacker wishes to perform a partial evaluation attack on two inputs m and m'. He computes encodings of $R_i \cdot P_m \cdot R_j^{-1}$ and $R_i \cdot P_{m'} \cdot R_j^{-1}$. Without the above defense mechanism, he could simply compare the two encodings by using the strategy outlined in Remark 3.2.3. If the $D_{i,b}$'s are used, then the random diagonal entries will with high probability prevent any comparison between $R_i \cdot P_m \cdot R_j^{-1}$ and $R_i \cdot P_{m'} \cdot R_j^{-1}$.

To see this, consider

$$R_i \cdot \begin{bmatrix} d_m & 0 \\ 0 & A_m \end{bmatrix} \cdot R_j^{-1} = \begin{bmatrix} R_{i,1} & R_{i,2} \\ R_{i,3} & R_{i,4} \end{bmatrix} \cdot \begin{bmatrix} d_m & 0 \\ 0 & A_m \end{bmatrix} \cdot \begin{bmatrix} R_{j,1} & R_{j,2} \\ R_{j,3} & R_{j,4} \end{bmatrix}$$

$$= \begin{bmatrix} R_{i,1} d_m R_{j,1} + R_{i,2} A_m R_{j,3} & R_{i,1} d_m R_{j,2} + R_{i,2} A_m R_{j,4} \\ R_{i,3} d_m R_{j,1} + R_{i,4} A_m R_{j,3} & R_{i,3} d_m R_{j,2} + R_{i,4} A_m R_{j,4} \end{bmatrix}.$$

In particular, every entry depends on d_m, so as long as m and m' use different matrices at some point during the computation (that is, as long as inp examines a bit where m and m' differ during one of the steps included in the partial evaluation), $d_m \neq d_{m'}$, so even if the bottom 5 by 5 corners of P_m and $P_{m'}$ match, it would be impossible to tell by examining the partial computations.

Remark 3.2.5 (Partial evaluations on similar inputs)**.** The goal of a partial evaluation attack is to determine whether two different inputs produce the same intermediate result at some point in the circuit. By examining inp, it is easy to construct two inputs m and m' such that this is the case. Any pair of inputs that agree on bits $inp(1), inp(2), \ldots, inp(k)$ will yield the same result in a partial evaluation of the first k steps in the branching program, regardless of the technique used to prevent partial evaluation attacks. Since learning that the partial evaluations on two different inputs are the same is not necessarily information accessible from an oracle to the obfuscated function, this is a breach of VBB security. However, it is not a breach of iO security.

In the indistinguishability setting, the goal of an attacker is to distinguish between the obfuscations of two functionally equivalent *input assignments* to a universal branching program. That is, two different inputs to the universal circuit that produce the same function. A user cannot hope to distinguish between two input assignments by only examining *inp*, since the universal branching program is oblivious. *inp* does not depend on the input, so it cannot help distinguish between two different inputs.

Preventing Mixed Inputs

There are many ways to defend obfuscation against mixed input attacks. Garg et al.'s original method is presented here.

Start by selecting random scalars $\alpha_{i,b} \in \mathbb{Z}_q$. Instead of randomizing and encoding the $D_{i,b}$'s, randomize and encode the matrices

$$\begin{bmatrix} d_{i,b} & 0 \\ 0 & \alpha_{i,b} A_{i,b} \end{bmatrix}.$$

If $\alpha_m = \prod_{i=1}^{n} \alpha_{i,m_{inp(i)}}$, then attempting to evaluate the resulting branching program on input m as before yields $\mathbf{s} \cdot \alpha_m A_m \cdot \mathbf{t}$, which is $\alpha_m \langle \mathbf{s}, \mathbf{t} \rangle$ if A_m is the identity. Unfortunately, it is not possible to compare this to $\langle \mathbf{s}, \mathbf{t} \rangle$ with the zero-testing parameter, and it is not possible to supply encodings of every possible $\alpha_m \langle \mathbf{s}, \mathbf{t} \rangle$ in the public parameters.

Instead, a "dummy" branching program is supplied in the public parameters. The dummy branching program consists of the matrices

$$\begin{bmatrix} d'_{i,b} & 0 \\ 0 & \alpha'_{i,b} I \end{bmatrix}.$$

It is randomized and encoded just as the "primal" branching program. Note that since all the $A_{i,b}$'s have been replaced by the identity matrix, the dummy program computes the constant function 1. Here, the $d'_{i,b}$'s are chosen randomly as they are in the primal program, but the $\alpha'_{i,b}$'s are chosen so that the output of the dummy program matches the output of the primal program on legal input assignments where the primal program would output 1.

Specifically, for each $j \le |m|$ and each $b \in \{0, 1\}$, choose random $\alpha'_{i,b}$'s such that

$$\prod_{inp(i)=j} \alpha_{i,b} = \prod_{inp(i)=j} \alpha'_{i,b}.$$

This restriction ensures that for legal input assignments, $\alpha_m = \alpha'_m$. Indeed if m is a legal input assignment, then

$$\alpha_m = \prod_{j=1}^{|m|} \prod_{inp(i)=j} \alpha_{i,m_i} = \prod_{j=1}^{|m|} \prod_{inp(i)=j} \alpha'_{i,m_j} = \alpha'_m.$$

However, if an attacker tries to use a mixed input assignment m, it is no longer guaranteed that

$$\prod_{inp(i)=j} \alpha_{i,m_i} = \prod_{inp(i)=j} \alpha'_{i,m_i},$$

because the $\alpha'_{i,b}$'s were selected only to ensure equality in the case where either all the m_i's are 1 or all the m_i's are 0.

Complete Garg et al. Obfuscation Algorithm

Combining all these components yields the complete obfuscation algorithm:

Given a circuit family $\mathcal{C}_\lambda$,

1. Find a universal circuit U_λ for the family $\mathcal{C}_\lambda$. Use Barrington's algorithm to convert U_λ into an equivalent oblivious matrix branching program $M = (I, A_1, \{A_{i,b}\}, inp)$. Also create a dummy branching program $M' = (I, A_1, \{I\}, inp)$ (note that both instances of inp are the same function).

2. Generate a GES with a zero-testing parameter at level-$\mathbf{1}^{n+2}$, where n is the length of M. Let *params* be the public parameters of the GES.

3. Choose a prime q such that messages in $\mathbb{Z}_q$ can be encoded by the GES. Choose four random 5-vectors with entries in $\mathbb{Z}_q$: $\mathbf{s}$, $\mathbf{t}$, $\mathbf{s}'$, $\mathbf{t}'$ along with four random m-vectors $\mathbf{r_s}$,$\mathbf{r_t}$,$\mathbf{r'_s}$, and $\mathbf{r'_t}$ for some fixed integer m. Also, choose two sets of random $(2m+5)\times(2m+5)$ invertible matrices: $\{R_i\}_{i=0}^n$ and $\{R'_i\}_{i=0}^n$. Choose two sets of $2m$-vectors with entries in $\mathbb{Z}_q$ to be the random diagonal entries: $\{d_{i,b}\}_{i=1}^n$ and $\{d'_{i,b}\}_{i=1}^n$. Finally, choose random scalars $\alpha_{i,b} \in \mathbb{Z}_q$ and $\alpha'_{i,b} \in \mathbb{Z}_q$ such that $\prod_{inp(i)=j} \alpha_{i,b} = \prod_{inp(i)=j} \alpha'_{i,b}$ for all j and $b \in \{0,1\}$.

4. Compute the randomized branching program and randomized dummy program: $M_R = (I, A_1, \{\tilde{A}_{i,b}\}, inp)$ and $M'_R = (I, A_1, \{\tilde{A'}_{i,b}\}, inp)$ where

$$\tilde{A}_{i,b} = R_{i-1} \cdot \begin{bmatrix} d_{i,b} & 0 \\ 0 & \alpha_{i,b}A_{i,b} \end{bmatrix} \cdot R_i^{-1}$$

and

$$\tilde{A}'_{i,b} = R'_{i-1} \cdot \begin{bmatrix} d'_{i,b} & 0 \\ 0 & \alpha'_{i,b} I \end{bmatrix} \cdot R'^{-1}_{i}.$$

5. Use the GES to generate level-$\mathbf{e_1}$ encodings $\hat{\mathbf{s}}$, $\hat{\mathbf{s}}'$ of $\mathbf{s}^* = (\mathbf{0}, \mathbf{r_s}, \mathbf{s}) \cdot R_0^{-1}$ and $\mathbf{s}^{*\prime} = (\mathbf{0}, \mathbf{r}'_{\mathbf{s}}, \mathbf{s}') \cdot R'_0$ respectively. Similarly, generate level-$\mathbf{e_{n+2}}$ encodings $\hat{\mathbf{t}}$, $\hat{\mathbf{t}}'$ of $\mathbf{t}^* = R_n \cdot (\mathbf{r_t}, 0, t)$ and $\mathbf{t}^{*\prime} = R'_n \cdot (\mathbf{r}'_{\mathbf{t}}, 0, t')$ respectively.

6. Encode the entries of each matrix $\tilde{A}_{i,b}$ and $\tilde{A}'_{i,b}$ at level $\mathbf{e_{i+1}}$. Call the encoded matrices $\hat{A}_{i,b}$ and $\hat{A}'_{i,b}$.

7. Let I_C be the set of steps in the universal branching program that examine an input bit corresponding to part of the circuit description C. Then the obfuscation of C is comprised of
$$\{(i, \hat{A}_{i,C_{inp(i)}}) \mid i \in I_C\}, \quad \{(i, \hat{A}'_{i,C_{inp(i)}}) \mid i \in I_C\},$$
$$\{(i, \hat{A}_{i,b}) \mid i \notin I_C, b \in \{0,1\}\}, \quad \{(i, \hat{A}'_{i,b}) \mid i \notin I_C, b \in \{0,1\}\},$$
$$\hat{\mathbf{s}}, \hat{\mathbf{t}}, \hat{\mathbf{s}}', \hat{\mathbf{t}}', params, inp.$$

To evaluate an obfuscation of C on the input j, let $J = (C, j)$ so that $U(J) = U(C, j) = C(j)$. Then use *params* to zero-test the matrix

$$(\hat{\mathbf{s}} \cdot \prod_{i=1}^{n} \hat{A}_{i,J_{inp(i)}} \cdot \hat{\mathbf{t}}) - (\hat{\mathbf{s}}' \cdot \prod_{i=1}^{n} \hat{A}'_{i,J_{inp(i)}} \cdot \hat{\mathbf{t}}').$$

If the zero-test returns true, then the primal and dummy produce the same result on the input j, and since the dummy program always computes I, the primal program also computes I. Therefore, $C(j) = 0$. Otherwise, $C(j) = 1$.

Remark 3.2.6 (Zero-izing attacks on obfuscation)**.** The Garg et al. indistinguishability obfuscator does not require evaluators to be able to generate their own encodings. All they need to evaluate obfuscated code is the zero testing parameter and the modulus of the multilinear map. As such, publishing level-$\mathbf{0}$ encodings to enable random sampling is unnecessary, as is publishing level-$\mathbf{1}$ encodings of $\mathbf{0}$ for re-randomization. This means that the zero-ization attacks from Chapter 2 do not apply in an obvious way to multilinear maps used in obfuscation.

Recall that these attacks depend on the attacker being able to construct low-level encodings of zero. In the multi-party Diffie Hellman procedure, low-level encodings of zero are published as part of the public parameters. In the obfuscation setting, no such encodings are published, and it is not clear how to construct them from the public data. Indeed, each matrix in the published branching program is an encoding of a matrix with entries randomized over $\mathbb{Z}_q$. Therefore, with high probability, none of the public matrices contain encodings of 0.

Researchers therefore remain hopeful that multilinear maps in the obfuscation setting are unaffected by the zero-ization attacks

3.2.2 Extending to Circuits in P

The paradigm of obfuscating circuits by converting them to equivalent branching programs is limited to obfuscating log depth circuits because the conversion process has an exponential dependency on the depth of the input circuit. Therefore, obfuscating larger circuits requires a different strategy. Garg et al. leverage Fully Homomorphic Encryption (FHE) and their NC^1 indistinguishability obfuscator to achieve obfuscation for P.

Assume the existence of an efficient FHE scheme with a decryption algorithm in NC^1. Also assume the existence of an indistinguishability obfuscator iO_{NC^1} for NC^1 circuits. To obfuscate a circuit C from a circuit family $\{\mathcal{C}_\lambda\}$, where $\{U_\lambda\}$ is a universal circuit family for $\{\mathcal{C}_\lambda\}$, one does the following:

1. Generate two FHE key pairs, (PK_1, SK_1) and (PK_2, SK_2).
2. Encrypt a description of C for input to U_λ under both public keys:

$$g_1 = Encrypt(PK_1, C),\ g_2 = Encrypt(PK_2, C).$$

3. Use iO_{NC^1} to obfuscate the program P_1. P_1 is the program that on input (m, e_1, e_2, ϕ) first checks to see if ϕ proves that e_1 and e_2 are the homomorphic evaluations of U_λ on the input C and m under the keys PK_1 and PK_2 respectively. If so, P_1 returns the decryption of e_1 under SK_1.
4. The obfuscation is $(g_1, g_2, PK_1, PK_2, P = iO_{NC^1}(P_1))$.

To evaluate on an input m,

1. Compute $e_1 = Eval(PK_1, U_\lambda(g_1, m))$ and $e_2 = Eval(PK_2, U_\lambda(g_2, m))$.

2. Compute a *low-depth* proof ϕ that e_1 and e_2 were computed correctly.

3. Evaluate $P(m, e_1, e_2, \phi)$.

Since the encryption scheme is fully homomorphic and g_1 is an encryption of C, $e_1 = \mathit{Encrypt}(PK_1, U_\lambda(C, m)) = \mathit{Encrypt}(PK_1, C(m))$, so the correct evaluation of the circuit is the decryption of e_1. Similarly for e_2.

Remark 3.2.7 (Two secret keys)**.** This extension algorithm looks bizarre because C gets encrypted by two different keys, but only one of the encryptions is actually published in obfuscated form. Why then is it necessary to include a second encryption in the public parameters?

The two key structure is a way to sidestep the drawback of indistinguishability obfuscation. The indistinguishability property of iO does not guarantee that $iO(P_1)$ conceals SK_1. However, $iO(P_2)$ where P_2 that is the same as P_1 except that it outputs the decryption of e_2 under SK_2 certainly conceals SK_1. P_2 contains no information about SK_1, and P_1 contains no information about SK_2. But P_1 and P_2 are functionally equivalent circuits, so $iO(P_1)$ is indistinguishable from $iO(P_2)$. Since $iO(P_1)$ is indistinguishable from $iO(P_2)$, and $iO(P_2)$ reveals nothing about SK_1, $iO(P_1)$ also reveals nothing about SK_1.

Step 2 is done by keeping track of the value of each wire when evaluating the circuit corresponding to $\mathit{Eval}(PK_1, U_\lambda(g_1, m))$. The program P_1 can check the validity of ϕ by checking the validity of each gate in the circuit $\mathit{Eval}(PK_1, U_\lambda(g_1, m))$ (since ϕ tells P_1 all the values of the intermediate wires). Crucially, this validation step is an NC^1 operation because checking correctness of a single gate is a constant depth circuit operation. Since the decryption algorithm of the FHE scheme is also assumed to be in NC^1, the program P_1 is in NC^1, and is therefore obfuscatable by iO_{NC^1}.

Example 3.2.8 (Low-depth proof)**.** Consider an AND circuit with two inputs and one output. There are three wires in this circuit, so the proof ϕ will be a 3-bit string. The verification circuit will have an input gate for each bit of the proof (3 input gates in this case). For each gate g_i in the input circuit, the verification circuit will have a gate G_i that computes the boolean statement $g_i(a, b) = c$ where a, b, and c are the bits of the proof that correspond to the input wires (a, b) and the output wire (c) of g_i.

For the AND circuit example, the verification circuit will have one equality test gate that will check whether ϕ_1 AND $\phi_2 = \phi_3$. Each gate in the verification circuit gets its input directly from an input gate corresponding to a particular wire in the input circuit. No gate in the verification circuit ever gets its input from another gate in the verification circuit. This means that each of the gates can be verified in constant depth independently of each other. All that is left for the verification circuit to do after it checks all the gates is check that all the outputs in the gate verification step are 1. This can be done in log depth, hence the verification circuit is in NC^1.

Security

Garg et al. prove their scheme is secure assuming the FHE scheme is IND-CPA, and that iO_{NC^1} is an indistinguishability obfuscator. They use the following sequence of hybrids to show that an obfuscation of C_0 is indistinguishable from an obfuscation of C_1 if C_0 and C_1 are functionally equivalent circuits:

1. $g_1 = \mathit{Encrypt}(PK_1, C_0)$, $g_2 = \mathit{Encrypt}(PK_2, C_0)$, and $P = iO_{NC^1}(P_1)$. This is an honest obfuscation of C_0.

2. $g_1 = \mathit{Encrypt}(PK_1, C_0)$, $g_2 = \mathit{Encrypt}(PK_2, C_1)$, and $P = iO_{NC^1}(P_1)$. Due to the IND-CPA property of the FHE scheme, this hybrid is indistinguishable from the first one. This follows because P_1 contains no information about SK_2, and therefore P contains no information that might be used to distinguish between $\mathit{Encrypt}(PK_2, C_0)$ and $\mathit{Encrypt}(PK_2, C_1)$. Note that the security of the FHE scheme is not enough to prove that game 2 is indistinguishable from game 5, because it might be the case that P does provide useful information for distinguishing between $\mathit{Encrypt}(PK_1, C_0)$ and $\mathit{Encrypt}(PK_1, C_1)$.

3. $g_1 = \mathit{Encrypt}(PK_1, C_0)$, $g_2 = \mathit{Encrypt}(PK_2, C_1)$, and $P = iO_{NC^1}(P_2)$. Here, P_2 is the same program as P_1, except that it outputs the decryption of e_2 under SK_2. Note that P_1 and P_2 compute the same function, because they both reject all inputs for which they would not return $\mathit{Decrypt}(\mathit{Encrypt}(U(C, m)))$. This hybrid is indistinguishable from the second by the indistinguishability of $iO_{NC^1}(P_1)$ and $iO_{NC^1}(P_2)$.

4. $g_1 = \mathit{Encrypt}(PK_1, C_1)$, $g_2 = \mathit{Encrypt}(PK_2, C_1)$, and $P = iO_{NC^1}(P_2)$. Due to the IND-CPA property of the FHE scheme, this hybrid is indistinguishable from the third one.

5. $g_1 = \mathit{Encrypt}(PK_1, C_1)$, $g_2 = \mathit{Encrypt}(PK_2, C_1)$, and $P = iO_{NC^1}(P_1)$. This is an honest obfuscation of C_1.

For more details, see Section 5.2 of [31].

3.3 Efficiency of Obfuscation

The Garg et al. indistinguishability obfuscator is hopelessly impractical. An obfuscation of a circuit with size s at security level λ has size $\mathcal{O}(\lambda^3 s^{10.92})$. To get a concrete idea of how large this is, consider the following example:

Example 3.3.1 (Obfuscation size for FHE decryption circuit)**.** Let the security level λ be 128 bits. A rudimentary lower bound on the depth of the circuit P_1 from Section 3.2.2 is 7. To see this, notice that P_1 is at least as deep as the underlying NC^1 FHE decryption algorithm. This decryption algorithm has depth logarithmic in λ (by the definition of NC^1), and $\log_2 \lambda = 7$.

Obfuscating P_1 requires finding a universal circuit U to which P_1 could be input. There is a construction of a universal circuit whose depth is not asymptotically larger than the depth of the input circuit [20]. That is, if P_1 has depth d and size λ, then a universal circuit $U_{\lambda,d}$ for circuits of size λ and depth d can be constructed such that the size of $U_{\lambda,d}$ is polynomial in λ, and the depth of $U_{\lambda,d}$ is $\mathcal{O}(d)$. However, for shallow circuits (such as $d = 7$), there is a large constant factor increase in depth.

If this step is ignored, and P_1 is instead converted directly to a branching program using Barrington's algorithm, the resulting branching program has length $n \approx 4^7 = 16384$.

Therefore, even with very generous assumptions about the length of the branching program to be obfuscated, the multilinear map needs to be given with multilinearity level 16384. Using appropriate CLT parameter sizes gives a modulus x_0 with size $\approx 2^{46}$ bits. The total obfuscation is about 65536 $(2m + 5) \times (2m + 5) = (4n + 15) \times (4n + 15)$ encoded matrices, which is roughly 2^{48} encodings. The total size of the obfuscation of P_1 is therefore about 2^{94} bits.

Example 3.3.2 (Obfuscation size for depth λ^3 circuit)**.** Suppose that for security level λ, we wish to obfuscate a circuit C_λ of input size λ, depth λ^3, and size λ^5. Clearly this circuit is not in NC^1, so we will use the bootstrapping method from Section 3.2.2.

Using the depth-universal circuit construction from [20] gives a universal circuit U_λ of depth $\mathcal{O}(\lambda^3)$ and size $\mathcal{O}(5\lambda^{18} \log \lambda)$. The circuit Φ_λ for checking a low-depth proof that U_λ was computed correctly has one input for each wire of U_λ, and depth on the order of the log of the number of inputs. So Φ_λ has depth $\mathcal{O}(18 \log \lambda)$. The program P_1 requires running Φ_λ, and then a FHE decryption circuit, which has depth $\mathcal{O}(\log \lambda)$. In total, P_1 has depth $d = \mathcal{O}(19 \log \lambda)$. Note that computing $iO_{NC^1}(P_1)$ requires first finding a universal circuit U_P for P_1. Recall that U_P can be constructed so that the depth of U_P is a constant factor times the depth of P_1. For the sake of simplicity, we will assume that the hidden constant is 1, so that the depth of U_P is $19 \log \lambda$.

Converting U_P into a branching program by Barrington's Theorem yields a branching program of length $n = 4^{19 \log \lambda} = \lambda^{38}$. The multilinearity required by the GES is therefore λ^{38}. The size of a single encoding for a GES with multilinearity λ^{38} is $\mathcal{O}(\lambda^{79})$. The obfuscated program will consist of $4n$ $(2m+5) \times (2m+5)$ matrices of encodings. Garg et al. suggest taking $m = 2n+5$ [31]. This gives a total of $4n \cdot (4n+15)^2 = \mathcal{O}(64n^3) = \mathcal{O}(64\lambda^{114})$ encodings.

The obfuscation size is therefore $\mathcal{O}(64\lambda^{193})$. For $\lambda = 128$, we get an obfuscation size of 2^{1357} bits. Note that this is much larger than the trivial VBB obfuscator of Bernstein et al. (see Example 3.0.6) which has size 2^{136} bits for this particular circuit.

Suffice it to say that some dramatic improvements are necessary.

3.3.1 Avoiding Barrington's Theorem

The first route researchers have taken towards practical obfuscation is to avoid Barrington's algorithm for converting circuits to branching programs. Barrington's algorithm outputs a branching program of length 4^d where d is the depth of the circuit to be converted. As seen in Example 3.3.1, this exponential dependency is a problem even for small circuits.

Barrington's algorithm can be avoided because programs can be represented by multiple branching programs. Instead of using Barrington's branching program to express a circuit, which is of length $n = 4^d$, one might like to find a shorter equivalent branching program to obfuscate. The first such method is due to Ananth et al. [1].

Instead of obfuscating an oblivious matrix branching program as in Definition 3.1.2, they obfuscate a "relaxed" branching program where the output is discerned by checking a particular entry of the branching program matrix product instead of checking whether the whole matrix is the identity. The relaxed branching program is much shorter, but no longer uses constant size matrices. This tradeoff between length and width produces a net efficiency gain because increasing the length increases the necessary level of multilinearity, but increasing the width does not. Since the high level of multilinearity is the main source of inefficiency, it is helpful to reduce the multilinearity even at the cost of increasing other parameters.

Furthermore, the width of branching programs in the Garg et al. obfuscator is increased to $4n + 15$ anyway as part of the defence against partial evaluation attacks.

Any NC^1 circuit of size s can be "balanced" to have depth at most $\approx 1.82 \log_2 s$ [43]. Using Barrington's algorithm to convert the formula to a branching program results in a branching program of length at most $n = 4^{1.82 \log_2 s} = s^{3.64}$ and width 5. Then $s^{3.64}$ levels of multilinearity are needed resulting in a CLT modulus of size $\mathcal{O}(4\lambda^3 s^{7.28})$. Ignoring the increase in width, the obfuscation consists of $2s^{3.64}$ encoded 5 by 5 matrices, so the total size of the obfuscation is $\mathcal{O}(\lambda^3 s^{10.92})$. Evaluating the obfuscated circuit requires multiplying $2s^{3.64}$ 5×5 matrices.

If we consider the matrices of the branching program to be $(4n+15) \times (4n+15)$ instead (see Section 3.2.1), we get a total obfuscation size of $\mathcal{O}(\lambda^3 s^{18.2})$.

Ananth et al. manage to convert a balanced circuit of size s (and depth $d \approx 1.82 \log_2 s$) instead to a relaxed branching program of length $s = 4^{d/3.64} \approx 1.46^d$ and width $2s$. In this case, only s levels of multilinearity are necessary, giving a CLT modulus of size $\mathcal{O}(4\lambda^2 s^2)$. The obfuscation consists of $2s$ encoded $2s \times 2s$ matrices, so the total size of the obfuscation is $\mathcal{O}(32\lambda^2 s^5)$. Evaluating the obfuscated circuit requires multiplying $2s$ $2s \times 2s$ matrices. Note that there is still an exponential dependency on the depth, but the base is significantly smaller.

Example 3.3.3 (Obfuscation size without Barrington)**.** Examples of FHE schemes with decryption algorithms in NC^1 have decryption algorithms of size $\tilde{O}(\lambda)$ and depth $\mathcal{O}(\log_2 \lambda)$ [17]. Therefore, a lower bound on the size of P_1 is λ. To obfuscate P_1 for $\lambda = 128$, Ananth's obfuscation algorithm requires multilinearity of at least 128 (instead of 16384 as in Example 3.3.1). This makes a CLT encoding about 7.46 billion bits, or 932 MB.

Evaluating the obfuscated program requires 128 multiplications of 30×30 matrices, where each entry is 932 MB. This is an improvement, but actually evaluating the program still requires $\approx 2^{58}$ operations (assuming integer multiplication of n-bit integers is done in $\mathcal{O}(n \log n \log \log n)$ and matrix multiplication of 30×30 matrices is done in $\approx 30^{2.373}$ multiplications).

3.3.2 Avoiding Branching Programs

Branching programs provide a nice structure for obfuscation, but there are two large efficiency problems. First, in both the Garg et al. and the Ananth et al. algorithms, a branching program derived from a circuit has length exponential in the circuit depth (4^d for the former, and 1.46^d for the latter). This means not only that the level of multilinearity depends exponentially on the depth, but that the time it takes to evaluate the obfuscated circuit depends exponentially on the depth. Second, branching programs are fundamentally limited to computing decision problems. Obfuscating a circuit with multiple bits of output using a branching program therefore requires one branching program per output bit.

Zimmerman describes an obfuscation algorithm that avoids branching programs completely [45]. Zimmerman's method eliminates the exponential dependency of evaluation time on the depth, and allows obfuscation of circuits with multiple output bits. Note that exponential multilinearity is still necessary.

Zimmerman's construction operates directly on keyed arithmetic circuits where the goal is to obfuscate the key, but not necessarily the structure of the circuit. Any circuit can be converted to a keyed circuit by using a universal circuit. Using this limitation allows for the gates of the circuit to be published so that the circuit can be evaluated without needing to be converted to a branching program. Multilinear encodings are used to encode the key, and enforce honest evaluation of the circuit on that key. For the complete construction, see [45].

The result is an obfuscation scheme where evaluation only takes $\mathcal{O}(d^2 s^2 + n^2)$ multilinear map operations where d is the circuit depth, s is the size, and n is the input size. Unfortunately, each gate of the circuit needs to be encoded at a separate level, and there are in general, exponentially many levels involved, which means the degree of multilinearity needed is therefore still exponential in d.

Chapter 4

Applications

Cryptographic multilinear maps and code obfuscation can be used to achieve a myriad of cryptographic goals. This chapter will be the subject of three such applications: broadcast encryption, non-interactive key exchange (NIKE), and RSA-FDH.

The goal of broadcast encryption (BE) is for a sender to be able to broadcast a message to a large set U of users, but only have a subset S, specified on sending, of them be able to decrypt. It should be the case that even if all the users outside S collude, they should not be able to discover the broadcast. A broadcast encryption scheme has a number of performance metrics including public key size, ciphertext size, and private key size. Ideally, key sizes and ciphertext size should not depend on the size of S. There is a simple broadcast encryption scheme where the public keys and ciphertexts grow linearly with the size of S, while the private keys are small. We will see how to use multilinear maps and code obfuscation to improve on this asymptotically [12]. We stress that these results are only an asymptotic improvement, and that currently, for realistic parameter sizes, existing broadcast encryption algorithms perform significantly better than their multilinear counterparts.

In Chapter 2, we saw that a one-round multi-party key exchange can be implemented using multilinear maps. However, this scheme requires a trusted third party (TTP) to generate keys. Furthermore, the TTP is trivially able to learn any user's private key. Boneh and Zhandry showed how to use indistinguishability obfuscation (instead of just multilinear maps) to create a non-interactive multi-party key exchange that does not require a trusted third party [14]. While it is clear that relying on iO instead of multilinear maps has a huge impact on performance, this remains an interesting result because of the avoidance of the TTP, and possible avoidance of the zero-izing attacks on the CLT multilinear map.

The security of RSA-FDH is easily proven in the random oracle model [5]. However, cryptographers have been unable to find a concrete hash function that behaves as a random oracle. In 2014, Hohenberger, Sahai and Waters showed how to use indistinguishability obfuscation to create user-specific hash functions for which it is possible to prove the security of RSA-FDH without the random oracle model [37]. Since the hash functions are user specific, and are included as part of a public key, this is not true RSA-FDH in the sense that it would not be possible to simply substitute this new iO hash function for the old hash function in an implementation of RSA-FDH. Nevertheless, this application is an excellent example of the power that indistinguishability obfuscation could one day bring to the world of cryptography.

4.1 Broadcast Encryption

Definition 4.1.1 (Broadcast Encryption)**.** Let G be a set of identities. A (multi-sender) *broadcast encryption* (BE) scheme consists of three efficient algorithms:

- $KeyGen(G, \lambda)$ takes the identity set and a security parameter λ. $KeyGen$ outputs a secret key sk_i and a public key pk_i for each $i \in G$.
- $Encrypt(S)$ takes $S \subset G$ and outputs the encryption c of a random key k for use in a predetermined symmetric encryption scheme, and a header h.
- $Decrypt(S, h, c, sk_i)$ outputs k if $i \in S$ and $c = Encrypt(S)$.

In a multi-sender broadcast encryption scheme, KeyGen is run by a trusted third party who distributes the key pairs (sk_i, pk_i) to each identity in G. Any identity in G should be able to establish a symmetric scheme key k among any subset $S \subset G$. It should be the case that even if every user outside of S colludes, they are not able to discover k. In a *single-sender* broadcast encryption scheme, only the central entity known as the "broadcaster" can encrypt messages.

The three main efficiency metrics used by broadcast encryption schemes are the ciphertext size, the public key size, and the private key size. Ideally, none of these should depend on the size of S, but finding a scheme where that is true has proven difficult.

4.1.1 Boneh-Silverberg Broadcast Encryption

Example 4.1.2 (Elementary broadcast encryption scheme)**.** Consider the following broadcast encryption scheme: Each user in G has a public-private key pair for some public key encryption scheme. An entity i that wishes to broadcast to a subset of users S encrypts a random key k for a symmetric scheme $|S|$ times, once using each public key of the users in S. While the size of each private key in this scheme does not depend on S, the size of the ciphertext grows linearly with the size of S. Similarly, the public parameters also grow linearly with the number of users. In light of this simple algorithm, the goal of broadcast encryption design is to reduce the size of the ciphertext overhead and the public parameters while keeping the private keys small.

Boneh and Silverberg showed that we can get a single-sender broadcast scheme from multilinear maps with no ciphertext overhead, where the private key is a single group element [10]. It goes as follows:

Let $e\colon \mathbb{G}^N \to \mathbb{G}_T$ be an ideal cryptographic multilinear map (see Definition 2.1.1).

1. KeyGen: Fix a function $F\colon \{0,1\}^m \to \mathbb{G}^N$. Choose a random seed $a \in \{0,1\}^m$, and set $F(a) = (g_1, \ldots, g_N)$. Finally, let $g \in_R \mathbb{G}$ be a random generator. Choose a random α. The sender's secret key is α, and user i's secret key is $sk_i = g_i^\alpha$.

2. Let
$$\phi_S(i) = \begin{cases} g_i, & \text{if } i \in S, \\ g, & \text{otherwise.} \end{cases}$$
$Encrypt(S) = e(\phi_S(1), \ldots, \phi_S(N))^\alpha$.

3. $Decrypt(S, g_i^\alpha) = e(\phi_S(1), \ldots, \phi_S(i-1), g_i^\alpha, \phi_S(i+1), \ldots, \phi_S(N))$.

Note that in this case, publishing $c = Encrypt(S)$ is unnecessary (and would in fact be equivalent to publishing the secret key k). Indeed, Decrypt only needs access to S, and sk_i and not c or h. As such, the header in this scheme has size 0, and the "ciphertexts" are the size of one element of $\mathbb{G}_T$. Furthermore, each private key is also only one element of $\mathbb{G}$. Finally, the public parameters are F and a. For security purposes, Boneh and Silverberg take m (the bit-length of a) to be super-linear in $\log \lambda$ so that neither F nor a depend on N [10]

Remark 4.1.3 (Boneh-Silverberg broadcast encryption scheme with CLT)**.** This scheme can be instantiated using CLT encodings. In the CLT analog, the g_i's should be level-1 encodings of messages m_i, and g should be a level-1 encoding of 1. Finally, the sender's secret key K should be a level-0 encoding of α, and each user's secret key should be a level-1 encoding of $m_i \cdot \alpha$. A user's secret key should *not* be $K \cdot g_i$ (even though this is a valid level-1 encoding of $m_i \cdot \alpha$) because this would allow every user to easily compute K. A user's secret must be some other level-1 encoding of $m_i \cdot \alpha$.

The problem with instantiating this scheme using CLT is that it needs multilinearity equal to the number of users. As we've seen, the size of each encoding is quadratic in the multilinearity necessary. So despite the fact that private keys are a single encoding, the size of this encoding scales linearly with the number of users. Similarly, the ciphertext is an encoding, and thus scales quadratically with the number of users. So in fact, this is much worse than the trivial scheme from Example 4.1.2.

4.1.2 Bilinear Broadcast Encryption

Before there were multilinear maps, Boneh, Gentry, and Waters (BGW) devised a multi-sender broadcast scheme that uses bilinear maps, where the ciphertext and private key sizes are constant in the number of users [8].

Let $e\colon \mathbb{G}_1^2 \to \mathbb{G}_T$ be a cryptographic bilinear pairing. Let G be a set of identities with size N. For simplicity, we assume that $G = \{1, 2, \ldots, N\}$.

1. *KeyGen*(G, λ): Select a random generator g of $\mathbb{G}$ and random integers α,γ. Denote $g_i = g^{\alpha^i}$ for $1 \leq i \leq 2N$. Identity i's public key is g_i, and their private key is g_i^{γ}. Additionally, g, g^{γ}, and g_i for $i \neq N+1$ are published as part of the public parameters.

2. *Encrypt*(S): Pick a random t. Let $h = (g^t, (g^{\gamma} \cdot \prod_{j \in S} g_{N+1-j})^t)$. The key K shared among users in S is $K = e(g_{N+1}, g)^t$. The sender can compute this by computing $e(g_N, g_1)^t$.

3. *Decrypt*(S, h, sk_i): If $h = (C_0, C_1)$, compute
$$K = \frac{e(g_i, C_1)}{e(g_i^{\gamma} \cdot \prod_{j \in S, j \neq i} g_{N+1-j+i}, C_0)}.$$

The private keys are each a single group element, and the ciphertext overhead is only two group elements. Unfortunately, the public key still scales linearly with the number of users.

4.1.3 Multilinear Broadcast Encryption

Boneh, Waters, Zhandry (BWZ) make a small modification of the bilinear broadcast scheme to reduce the public key size [12]. Their basic idea is to use the same scheme, but use asymmetric multilinear maps to compress the public key.

Let G be a set of identities with $|G| = N = 2^n - 1$. Let e_i be the i'th characteristic vector in $\mathbb{Z}^n$. Finally, let e be an asymmetric multilinear map with base-level encodings at level-e_i, and a zero-test parameter at level-$\mathbf{2}$ (the all-2's vector). Let p be a large prime for which elements of $\mathbb{Z}_p$ can be encoded under the asymmetric multilinear map.

1. $KeyGen(G, \lambda)$: For fixed random α, $\gamma \in \mathbb{Z}_p$, let X_i be a level-e_i encoding of α^{2^i} for $i < n$, and let X_n be a level-$\mathbf{1}$ encoding of α^{2^n+1}. Also, let Y be a level-$\mathbf{1}$ encoding of γ, and W be a level-$\mathbf{2}$ encoding of α^{2^n}. The public key consists of the public parameters to e, W, X_i for $i \leq n$, and Y. Identity i's private key sk_i is a level-$\mathbf{1}$ encoding of $\alpha^i \cdot \gamma$.

2. *Encrypt*(S): Sample a random level-0 encoding c. Suppose that c is an encoding of t. Let
$$h = (c_1, c \cdot (Y + \sum_{i \in S} Z_{2^n - i}))$$
where c_1 is a level-$\mathbf{1}$ encoding of t, and Z_j is a level-$\mathbf{1}$ encoding of α^j. The key K shared among users in S will be $K = W \cdot c$, which is a level-$\mathbf{2}$ encoding of $t \cdot \alpha^{2^n}$.

   Z_j can easily be computed from the X_i's as follows. Let $j_1 j_2 \dots j_n$ be the binary representation of j. Let
$$X_i' = \begin{cases} X_i, & j_i = 1, \\ V_i, & j_i = 0, \end{cases}$$
where V_i is a level-e_i encoding of 1. Then
$$Z_j = \prod_{i=1}^{n} X_i'$$
is a level-$\mathbf{1}$ encoding of
$$\prod_{i=1}^{n} \alpha^{j_i \cdot 2^i} = \alpha^{\sum_{i=1}^{n} j_i \cdot 2^i} = \alpha^j$$
as required.

3. *Decrypt*(S, h, sk_i): If $h = (C_0, C_1)$, compute

$$K = (Z_i \cdot C_1) - \left((sk_i + \sum_{j \in S, j \neq i} Z_{2^n - j + i}) \cdot C_0 \right).$$

Correctness

To prove correctness, it suffices to show that the output of *Decrypt*(S, h, sk_i) is a level-**2** encoding of $t \cdot \alpha^{2^n}$. To get the canonical information from this key (as K will be a *different* encoding of $t \cdot \alpha^{2^n}$ for each user), each user must apply the zero-testing process.

Now, $Z_i \cdot C_1$ is a level-**2** encoding of $\alpha^i \cdot t \cdot \left(\gamma + \sum_{j \in S} \alpha^{2^n - j} \right)$, and

$$\left((sk_i + \sum_{j \in S, j \neq i} Z_{2^n - j + i}) \cdot C_0 \right)$$

is a level-**2** encoding of $t \cdot \left(\alpha^i \cdot \gamma + \sum_{j \in S, j \neq i} \alpha^{2^n - j + i} \right)$. So the output of *Decrypt* is a level-**2** encoding of

$$t \cdot \sum_{j \in S} \alpha^{2^n - j + i} - t \cdot \sum_{i \in S, j \neq i} \alpha^{2^n - j + i} = t \cdot \alpha^{2^n}$$

as required.

Efficiency

Compared to the bilinear scheme, the ciphertext overhead is still two "group elements" (though, the group is different), and the private keys are still one group element. However, the number of group elements that need to be published in the public parameters is now only logarithmically dependent on N instead of linearly. This comes at the cost of an extra computational step. The real cost however is the use of a multilinear map. While the public parameters may be very few group elements, those group elements are much larger than they were in the bilinear case.

Example 4.1.4 (Comparison with bilinear broadcast encryption)**.** If the size of the identity set G is $N = 2^n - 1$, the BWZ broadcast scheme requires a $\log_2 N$-linear map. Recall from

Section 2.3.2 that the public parameters of such a multilinear map (when sampling and re-randomization are required) consist of approximately $2\lambda + 2\lambda\sqrt{2\log_2 N \cdot}$ encodings. There are $\log_2 N$ additional public encodings in the BWZ scheme, making for a total of

$$\log_2 N + 2\lambda + 2\lambda\sqrt{2\log_2 N}$$

encodings. Each encoding is $\mathcal{O}(4\lambda^3 \cdot (\log_2 N)^2)$ bits.

For $\lambda = 128$, we therefore have that the total size of the public parameters in the BWZ scheme is

$$8,388,608 \cdot ((\log_2 N)^3 + 256(\log_2 N)^2 + 256(\log_2 N)^2\sqrt{2\log_2 N}) \text{ bits.}$$

The bilinear broadcast scheme on the other hand has N public keys, each of which is a single group element. For security level $\lambda = 128$, a group element in a bilinear pairing should be 256 bits. The public key size is therefore about $256 \cdot N$ bits.

Therefore, with the current state of multilinear maps, the public parameters of the bilinear broadcast scheme are smaller than those of the multilinear scheme for all sets of identities less than about 109 billion.

Additionally, since encodings are $\mathcal{O}(4\lambda^3 \cdot (\log_2 N)^2)$ bits, compared to 2λ bits in the bilinear scheme, encryption and decryption *time* now depend on N. In particular, multiplying two 128-bit secure encodings takes at least $32,768 \cdot (\log_2 N)^2$ times longer than multiplying two elliptic curve points.

Remark 4.1.5 (Computing X_i)**.** Computing X_i in general requires knowledge of the multilinear map secrets. Without the secrets, one would try and compute X_i by first sampling a random encoding to select α (note that this sampling procedure would not actually reveal α), and then by squaring this encoding an appropriate number of times. Unfortunately current multilinear maps are bounded in the number of multiplications they can perform.

It is easy to convert the BWZ multilinear scheme into a single-sender system. Instead of having a TTP generate the multilinear map parameters, simply have them be generated by the sender. Also, do not publish any sampling or re-randomization encodings in the public parameters. This way, only the sender who knows the multilinear map secrets will be able to generate encodings, and thus only the entity that knows the multilinear map secrets can generate valid headers. Doing this also reduces the size of the public parameters by eliminating the need to publish sampling and re-randomization encodings.

4.2 NIKE

The remaining sections discuss applications of iO. When working with VBB obfuscation, the benefit that obfuscation provides is usually pretty clear. For example, when converting a symmetric key encryption scheme to a public key encryption scheme, VBB($Encrypt(k, m)$) acts as a black box encryption oracle. An attacker can learn nothing of k from having an encryption oracle, and it is therefore safe to publish the black box obfuscation of $Encrypt(k, m)$. However, it is not obvious that an indistinguishability obfuscator can be used in the same way. Indistinguishability obfuscations can leak information about the plaintext circuit, as long the obfuscation of every other functionally equivalent circuit of the same size reveals the same information. Then perhaps it is not safe to publish $C = iO(Encrypt(k, m))$ because C, and all other obfuscations of circuits that compute the function $Encrypt(k, m)$, might leak information about k.

The overall strategy when trying to obfuscate a circuit P with iO is to use the indistinguishability property, along with additional assumptions to show that there exists a circuit C such that

1. $iO(C)$ is indistinguishable from $iO(P)$; and
2. C reveals nothing about P.

Note that it is enough for C merely to *exist*. It need never be computed. As long as there exists a circuit whose obfuscation is both indistinguishable from $iO(P)$ and reveals nothing about P, $iO(P)$ will reveal nothing of P.

C is often what's called a *punctured* version of P. What this means is that C is equal to P on all but polynomially many inputs, where C appears random instead. Once an attacker commits to an input m, P can be punctured at m to get C_m such that $C_m(n) = P(n)$ for all $n \neq m$, and $C_m(m)$ is random. The result is that C_m contains all the same information as P, except the information the attacker is interested in. If it can be proven that $iO(C_m)$ is indistinguishable from $iO(P)$ (which requires more than just the iO indistinguishability property since C_m and P compute different circuits), then that will be enough to prove that $iO(P)$ reveals "nothing" about P, since it will be indistinguishable from a different obfuscation that provably "reveals nothing" about P.

This section focuses on a particular example of this general technique, and demonstrates how the preceding strategy is formalized.

4.2.1 PRGs and Punctured PRFs

The Boneh-Zhandry NIKE makes use of three cryptographic primitives: pseudorandom generators (PRG), punctured pseudorandom functions (punctured PRF), and iO. A PRG is a function that "extends" a random input to a longer pseudorandom output. A punctured PRF is a PRF where knowledge of the punctured key enables the computation of the PRF at all but a polynomial set of points.

Definition 4.2.1. A *pseudorandom generator* PRG is a function $\mathrm{PRG} : \{0,1\}^n \to \{0,1\}^m$ where $m > n$ such that no polynomial time distinguisher $\mathcal{D}$ has non-negligible advantage in the following security game:

1. The challenger chooses $s \in_R \{0,1\}^n$ and $b \in_R \{0,1\}$.
2. $\mathcal{D}$ is given $r_b = \begin{cases} \mathrm{PRG}(s), & b = 0, \\ r \in_R \{0,1\}^m, & b = 1. \end{cases}$
3. $\mathcal{D}$ returns $b' \in \{0,1\}$.

$\mathcal{D}$'s *advantage* is $|P(b' = 1 \mid b = 1) - P(b' = 0 \mid b = 1)|$.

Definition 4.2.2. Let R denote a random function from $\{0,1\}^n$ to $\{0,1\}^m$. A *PRF* is a function $PRF : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^m$ where for $K \in_R \{0,1\}^k$, no polynomial time attacker can distinguish between $PRF(K, \cdot)$ and $R(\cdot)$ with polynomially many queries to the oracle functions.

Definition 4.2.3. A *puncturable PRF* is a PRF together with a function **Puncture** that takes a PRF key K and a polynomial-size set $S \subset \{0,1\}^n$. **Puncture** outputs a *punctured PRF* PRF_S and a punctured PRF key K_S such that $PRF_S(K_S, x) = PRF(K, x)$ for all $x \notin S$, and no polynomial time adversary $\mathcal{A}$ has non-negligible advantage in the following security game:

1. $\mathcal{A}$ chooses a set $S \subset \{0,1\}^n$, and asks the challenger for a PRF punctured at S.
2. The challenger chooses $K \in_R \{0,1\}^k$, and computes

   $$\mathbf{Puncture}_{PRF}(K, S) = (PRF_S, K_S).$$

   PRF_S and K_S are given to $\mathcal{A}$.

3. The challenger chooses $b \in_R \{0,1\}$.
4. $\mathcal{A}$ makes polynomially many queries to an oracle $PRF(K,\cdot)$.
5. $\mathcal{A}$ asks the challenger for polynomially many

$$r_b = \begin{cases} PRF(K,x), & b=0, \\ r \in_R \{0,1\}^m, & b=1, \end{cases}$$

   such that $x \in S$, and x was not queried in step 4.
6. $\mathcal{A}$ returns $b' \in \{0,1\}$.

$\mathcal{A}$'s *advantage* is $|P(b'=1 \mid b=1) - P(b'=0 \mid b=1)|$.

As shown by Boneh and Waters, puncturable PRFs can be easily constructed [11].

4.2.2 Boneh and Zhandry's NIKE

Let G be a set of identities (users), an arbitrary subset S of which may wish to compute a shared secret. To do so, each party $i \in S$ chooses a secret key k_i for a puncturable pseudorandom function PRF and a secret random seed $s_i \in \{0,1\}^\lambda$ for a pseudorandom generator $\mathrm{PRG} : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$, and computes $x_i = \mathrm{PRG}(s_i)$. Finally, a public random $x_0 \in \{0,1\}^{2\lambda}$ is agreed upon. Consider the program PK_i created by party i:

- Inputs: $j \in G$, $s \in \{0,1\}^\lambda$, $x_1, \ldots, x_{|G|} \in \{0,1\}^{2\lambda}$
- Constants: k_i

    1. If $x_j \neq \mathrm{PRG}(s)$ return null.
    2. Output $PRF_{k_i}(x_1, \ldots, x_{|G|})$.

User i's public key consists of x_i and $iO(PK_i)$. A user i in S can compute the shared secret by first computing for each $j \in G$

$$\hat{x}_j = \begin{cases} x_j, & j \in S, \\ x_0, & j \notin S. \end{cases}$$

Next they compute $iO(PK_{i^*})(i, s_i, \hat{x}_1, \ldots, \hat{x}_{|G|}) = PRF_{k_{i^*}}(\hat{x}_1, \ldots, \hat{x}_{|G|})$ where i^* is the smallest element of S.

The only way for an attacker to provide $iO(PK_{i^*})$ with input that does not abort is to come up with a seed and corresponding pseudorandom number. But any seed they choose that is not equal to one of the s_i for $i \in S$ will alter the input to $PRF_{k_{i^*}}$, and hence fail to compute the shared secret.

Definition 4.2.4 (Static security)**.** The static security game for a NIKE is as follows:

A PPT adversary $\mathcal{A}$ selects a subset S of G. For each $i \in S$, choose a random seed $s_i \in \{0,1\}^\lambda$. Compute $x_i = \mathrm{PRG}(s_i)$ and $iO(PK_{i^*})$. Finally, choose $b \in \{0,1\}$ at random and compute

$$z_b = \begin{cases} PRF_{k_{i^*}}(\hat{x}_1, \ldots, \hat{x}_{|G|}), & b = 0, \\ r, & b = 1, \end{cases}$$

where r is chosen randomly. The adversary is given the x_i's, $iO(PK_{i^*})$, z_b, and x_0. The adversary returns $b' \in \{0,1\}$, and its advantage is $|P(b' = 1 \mid b = 1) - P(b' = 1 \mid b = 0)|$.

Theorem 4.2.5 (Boneh-Zhandry)**.** If PRG is a secure pseudorandom generator, PRF is a secure puncturable pseudorandom function, and iO is an indistinguishability obfuscator, then the Boneh-Zhandry NIKE is a statically secure NIKE.

Proof. Let **Game 0** be the static NIKE security game. For each $j \leq |G|$ let **Game $\mathbf{0_j}$** be the same as **Game 0**, but for all $i \leq j$, instead of computing $x_i = PRG(s_i)$, the challenger chooses x_i at random from $\{0,1\}^{2\lambda}$. Also, let **Game 1** be **Game $\mathbf{0_{|G|}}$**. If $adv_j(\mathcal{A})$ is the advantage of $\mathcal{A}$ in **Game $\mathbf{0_j}$**, and PRG is a secure pseudorandom generator, $adv_j(\mathcal{A}) \approx adv_{j+1}(\mathcal{A})$.

To see this, construct a PRG distinguisher $\mathcal{D}$:

1. Run $\mathcal{A}$ to get S.

2. The PRG challenger fixes a $b \in \{0,1\}$ and $\mathcal{D}$ is given

$$r_b = \begin{cases} \mathrm{PRG}(s), & b = 0, \\ R, & b = 1, \end{cases}$$

   where R is chosen randomly. $\mathcal{D}$'s task is to find b.

3. For $i > j+1$, select random $s_i \in \{0,1\}^\lambda$ and compute $x_i = \mathrm{PRG}(s_i)$. Let $x_{j+1} = r_b$. Finally, let the remaining x_i's be chosen randomly.

4. Send z_0, $iO(PK_{i^*})$ and the x_i's to $\mathcal{A}$.

5. If $\mathcal{A}$ returns 0, return 0; otherwise, return 1.

$\mathcal{D}$'s advantage is $|P(D \to 0 \mid b = 0) - P(D \to 0 \mid b = 1)|$. But if $b = 0$, this is exactly **Game $\mathbf{0_j}$**, and if $b = 1$, this is exactly **Game $\mathbf{0_{j+1}}$**, so $\mathcal{D}$'s advantage is $|adv_j(\mathcal{A}) - adv_{j+1}(\mathcal{A})|$. But since PRG is a secure pseudorandom generator, $\mathcal{D}$'s advantage is negligible, hence $adv_j(\mathcal{A}) \approx adv_{j+1}(\mathcal{A})$. In particular, **Game 0** is indistinguishable from **Game 1**.

Let **Game 2** be the same as **Game 1**, except instead of obfuscating PK_{i^*}, the challenger obfuscates PK'_{i^*} which is the same program as PK_{i^*}, except that it computes PRF_{i^*} using the key punctured at $(\hat{x}_1, \ldots, \hat{x}_{|G|})$. If iO is an indistinguishability obfuscator, then the advantage of $\mathcal{A}$ in **Game 1** is negligibly close to the advantage of $\mathcal{A}$ in **Game 2**.

To see this, construct an iO distinguisher $\mathcal{D}_{iO}$ as follows:

1. Run $\mathcal{A}$ to get S.

2. For each $i \in S$, choose x_i randomly. Compute $C_0 = PK_{i^*}$ and $C_1 = PK'_{i^*}$. Send C_0 and C_1 to the iO challenger.

3. The iO challenger returns $iO(C_b)$ for some randomly chosen $b \in \{0, 1\}$. $\mathcal{D}_{iO}$'s goal is to find b.

4. Compute $z_0 = PRF_{i^*}(\hat{x}_1, \ldots, \hat{x}_{|G|})$. Give $\mathcal{A}$ $iO(C_b)$, the x_i's, and z_0.

5. If $\mathcal{A}$ returns 0, return 0, otherwise return 1.

Note that PK_{i^*} and PK'_{i^*} are functionally equivalent on all inputs except possibly at $(\hat{x}_1, \ldots, \hat{x}_{|G|})$. However, with high probability, none of the x_i's have PRG pre-images (since they were all chosen randomly from a set of size $2^{2\lambda}$, and there are only 2^λ possible preimages). In particular, with overwhelming probability, both PK_{i^*} and PK'_{i^*} return null on the only input they could possibly be different. Therefore, they compute the same function, and so by the security of iO, $iO(PK_{i^*})$ and $iO(PK'_{i^*})$ are indistinguishable.

So $\mathcal{D}_{iO}$ has negligible advantage. But the advantage of $\mathcal{D}_{iO}$ is $|P(\mathcal{D}_{iO} \to 0 \mid b = 0) - P(\mathcal{D}_{iO} \to 0 \mid b = 1)|$. If $b = 0$, we are in **Game 1**, and if $b = 1$, we are in **Game 2**. So the advantage of $\mathcal{D}$ is $|adv_1(\mathcal{A}) - adv_2(\mathcal{A})|$, which is negligible. Hence $adv_1(\mathcal{A}) \approx adv_2(\mathcal{A})$

But no PPT adversary $\mathcal{A}$ can have significant advantage in **Game 2** as long as PRF'_{i^*} is a secure puncturable PRF. Suppose there exists an $\mathcal{A}$ with significant advantage in **Game 2**. Construct a punctured PRF adversary $\mathcal{A}_p$ as follows:

1. Run $\mathcal{A}$ to get S.

2. For each $i \in S$, choose x_i at random. Ask the $\mathcal{A}_p$ challenger for a PRF punctured at $(\hat{x}_1, \ldots, \hat{x}_{|G|})$. Call it PRF'_{i^*}.

3. Since there is only one element of the punctured set S (namely $\hat{x}_1, \ldots, \hat{x}_{|G|}$), $\mathcal{A}_p$ does not need an oracle to make PRF queries (PRF_{i^*} can find $PRF(K, x)$ for all $x \notin S$). Furthermore, since S has size 1, $\mathcal{A}_p$ has only one choice for which seed to submit to the challenger for distinguishing.

4. The $\mathcal{A}_p$ challenger chooses a random $b \in \{0, 1\}$ and gives

$$z_b = \begin{cases} PRF_{i^*}(\hat{x}_1, \ldots, \hat{x}_{|G|}), & b = 0, \\ r, & b = 1, \end{cases}$$

where r is chosen randomly. $\mathcal{A}_p$'s task is to find b.

5. Compute $iO(PK'_{i^*})$ using PRF'_{i^*}. Send the x_i's, $iO(PK'_{i^*})$, and z_b to $\mathcal{A}$.

6. If $\mathcal{A}$ outputs 0, output 0. Otherwise return 1.

Since $\mathcal{A}_p$ has negligible advantage, $\mathcal{A}$ must also have negligible advantage in **Game 2**. Therefore $\mathcal{A}$ also has negligible advantage in **Game 0**, and so no PPT $\mathcal{A}$ exists that wins the static NIKE game with significant advantage. Hence this construction is a statically secure NIKE. □

4.3 RSA-FDH

RSA Full Domain Hash (RSA-FDH) is a simple RSA-based signature scheme that can easily be proven secure in the random oracle model.

Example 4.3.1 (RSA-FDH)**.** Let N be an RSA modulus, and let (e, d) be an RSA public-private key pair. Let $h\colon \{0,1\}^* \to \mathbb{Z}_N$ be a random function with image $\mathbb{Z}_N$. To sign a message $M \in \{0,1\}^*$, compute $\sigma = h(M)^d \mod N$. To verify a signature σ on a message M, check if $h(M) = \sigma^e \mod N$.

Assuming the RSA problem (RSAP) is hard, this scheme can be proven to be existentially unforgeable under adaptive chosen message attacks. Intuitively, if h is random, forging a signature on a message M' amounts to computing e^{th} roots of random elements of $\mathbb{Z}_N$. The fact that $h(M')$ is random renders the adaptive power of an adversary useless. It does not matter which message they choose to forge, because the randomness of h guarantees $h(M')$ is independent of all other information the adversary has learned. Hence they can gain no advantage by carefully selecting M'.

The use of the random oracle model to prove security is contentious. There are contrived signature schemes that can be proven secure in the random oracle model but fall to simple practical attacks [18]. For this reason, there is a desire for a signature scheme that is provably secure without the random oracle model. Ideally we would want to find a function h that behaves like a random oracle, so that implementations of RSA-FDH could simply substitute instances of the old hash function for the new one. Hohenberger, Sahai, and Waters (HSW) claim to have accomplished this using iO [37]. (Although we shall see that they have only come very close: the hash function they construct depends on the RSA public key. Therefore, each user needs their own hash function to publish as part of their public key.)

This section presents HSW's selectively secure signature scheme. They make some further modifications to arrive at an adaptively secure signature scheme. For the adaptive security construction, see [37].

Definition 4.3.2 (HSW Signature Scheme)**.** The HSW signature scheme is a set of three polynomial-time algorithms:

- KeyGen: Choose an RSA modulus and key pair N, (e, d). Choose a collision resistant hash function $h\colon \{0,1\}^* \to \mathbb{Z}_N$. Finally, choose a puncturable PRF (see Section 4.2.1) $F\colon \{0,1\}^k \times \mathbb{Z}_N \to \mathbb{Z}_N$ and a key $K \in_R \{0,1\}^k$ for F.

  Let FDH(M) be the circuit that first computes $m = h(M)$, and returns $F(K, m)^e \mod N$. Suppose also that FDH is padded appropriately (see proof for details). Let the hash function H be iO(FDH).

- Sign: A signature on the message M is computed as $\sigma = F(K, M)$; note that $\sigma = H(M)^d \mod N$.

- Verify: Given (M, σ, H, N, e), check if $\sigma^e \equiv H(M) \mod N$.

Theorem 4.3.3 (HSW)**.** If iO is iO-secure, F is a secure puncturable PRF, and RSAP is computationally infeasible, then the HSW signature scheme is selectively secure.

Proof. This theorem is proven in a sequence of three games. The first game comparison uses the iO security property. The second game comparison uses the puncturable PRF security property. Finally, the last game is proven impossible to win using the hardness of RSAP. **Game 0** is simply the selective security game for this signature scheme:

Definition 4.3.4 (Selective Security)**.** A signature scheme is *selectively secure* if no PPT adversary $\mathcal{A}$ has non-negligible advantage in the following security game:

1. $\mathcal{A}$ selects a message M^* to attempt to forge a signature σ^* on.
2. The challenger executes the KeyGen algorithm, and returns to $\mathcal{A}$ an RSA modulus N, an RSA public key e, and a hash function H.
3. $\mathcal{A}$ makes polynomially many queries to a signing oracle to acquire signatures for messages $M \neq M^*$.
4. $\mathcal{A}$ outputs σ^*.

$\mathcal{A}$'s advantage is $P(\text{Verify}(\sigma^*, M^*) = 1)$.

Game 1 is the same as **Game 0**, except that instead of responding to $\mathcal{A}$ with H, the challenger gives H^* to $\mathcal{A}$, where H^* is a "punctured" version of H:

For a fixed M^*, the challenger first computes $m^* = h(M^*)$, a PRF key $K^* = K(m^*)$ punctured at m^*, and $Z^* = F(K, m^*)^e \mod N$. Let $FDH^*(M)$ be the circuit that first computes $m = h(M)$, and then checks if m is equal to the hard-coded value m^*. If so, FDH^* returns the hard-coded value of Z^*. Otherwise, it returns $F(K^*, m)^e \mod N$.

Then $H^* = iO(\text{FDH}^*)$. Note that FDH and FDH* compute the same circuit, so H and H^* are indistinguishable, by the iO-security property.

Let $adv_0(\mathcal{A})$ and $adv_1(\mathcal{A})$ be the advantage of $\mathcal{A}$ in games 0 and 1 respectively. If iO is iO-secure, and FDH is padded to have the same length as FDH*, then $adv_0(\mathcal{A}) \approx adv_1(\mathcal{A})$. To see this, use $\mathcal{A}$ to construct an iO distinguisher $\mathcal{D}$ as follows:

1. Run $\mathcal{A}$ to get M^*.
2. Run KeyGen honestly to get N, (e, d), F, and K. Compute $m^* = h(M^*)$. Compute $K^* = K(m^*)$ as the PRF key K punctured at m^*. Finally, compute $z^* = F(K, m^*)^e \mod N$.
3. Let $C_0 = \text{FDH}$ and let $C_1 = \text{FDH}^*$ (where FDH is padded to the length of FDH*). Send C_0 and C_1 to the iO challenger.

4. The iO challenger randomly selects $b \in_R \{0, 1\}$, and returns $H' = iO(C_b)$.

5. Give $\mathcal{A}$ the input (N, e, H').

6. Respond to $\mathcal{A}$'s signature query on $M \neq M^*$ with $F(K, m)$.

7. If $\mathcal{A}$ outputs a valid signature σ^* on m^*, $\mathcal{D}$ outputs 1, otherwise it outputs 0.

$\mathcal{D}$'s advantage is $|P(\mathcal{D} \to 1 \mid b = 0) - P(\mathcal{D} \to 1 \mid b = 1)|$. But if $b = 0$, we are in **Game 0**, and if $b = 1$, we are in **Game 1**. Therefore, $\mathcal{D}$'s advantage is $|adv_0(\mathcal{A}) - adv_1(\mathcal{A})|$. But $\mathcal{D}$'s advantage is negligible, since no iO distinguisher can have significant advantage. Hence $adv_0(\mathcal{A}) \approx adv_1(\mathcal{A})$.

Game 2 is the same as **Game 1**, except that instead of responding to $\mathcal{A}$ with H^*, the challenger gives $\hat{H}$ to $\mathcal{A}$, where $\hat{H}$ randomly selects the hash at the point M^* instead of computing it honestly:

For a fixed M^*, the challenger computes $m^* = h(M^*)$, a PRF key $K^* = K(\{m^*\})$ punctured at m^*, and selects $t \in_R \mathbb{Z}_N$. The challenger computes $\hat{Z} = t^e \mod N$. Let $\hat{\text{FDH}}(M)$ be the circuit that first computes $m = h(M)$, and then checks if m is equal to the hard-coded value m^*. If so, $\hat{\text{FDH}}$ returns the hard-coded value of $\hat{Z}$. Otherwise, it returns $F(K^*, m)^e \mod N$.

Then $\hat{H} = iO(\hat{\text{FDH}})$. Note that $\hat{\text{FDH}}$ and FDH* do not compute the same function, but their obfuscations will nevertheless be indistinguishable because of the punctured PRF security of F. Intuitively, the plaintext circuit $\hat{\text{FDH}}$ reveals "nothing" about how to forge a signature on the fixed message M^*. Since iO(FDH) is indistinguishable from $iO(\hat{\text{FDH}})$, iO(FDH) must also reveal "nothing" about how to forge a signature on M^*.

To see that **Game 1** and **Game 2** are indistinguishable, construct an F-adversary $\mathcal{A}^F$ as follows:

1. Run $\mathcal{A}$ to get M^*. Send m^* to the PRF challenger.

2. The PRF challenger selects $b \in_R \{0, 1\}$ and K, and returns $K^* = K(\{m^*\})$ and

$$t_b = \begin{cases} F(K, m^*), & b = 0, \\ t \in_R \mathbb{Z}_N, & b = 1. \end{cases}$$

$\mathcal{A}^F$'s goal is to determine b.

3. Select N, (e, d), and construct $H' = iO(\text{FDH}')$. Here, $\text{FDH}'(M)$ is the circuit that computes $m = h(M)$, and then checks if m is equal to the hard-coded value m^*. If so, it returns $t_b^e \mod N$. Otherwise, it returns $F(K^*, m)^e \mod N$. Give N, e, and H' to $\mathcal{A}$.

4. Respond to $\mathcal{A}$'s signature queries on $M \neq M^*$ by computing $F(K^*, M)^e \mod N$.

5. $\mathcal{A}^F$ outputs 1 if $\mathcal{A}$ outputs a valid signature on m^*, and 0 otherwise.

We have $adv(\mathcal{A}^F) = |P(\mathcal{A}^F \to 1 \mid b = 1) - P(\mathcal{A}^F \to 1 \mid b = 0)|$. If $b = 0$ then $t_b = F(K, m^*)$, and so $\text{FDH}' = \text{FDH}^*$; thus we are in **Game 1**. If $b = 1$ then $t_b = t$, and so $\text{FDH}' = \hat{\text{FDH}}$; so we are in **Game 2**. Therefore, $adv(\mathcal{A}^F) = |adv_1(\mathcal{A}) - adv_2(\mathcal{A})|$. But the advantage of $\mathcal{A}^F$ is negligible by the assumption that F is a secure puncturable PRF. Therefore $adv_1(\mathcal{A}) \approx adv_2(\mathcal{A})$.

Finally we prove that if RSAP is hard, then $adv_2(\mathcal{A})$ is negligible. Construct an RSAP solver $\mathcal{A}^R$ as follows:

1. $\mathcal{A}^R$ is given (N, e) and $z^* \in_R \mathbb{Z}_N$ where $z^* \equiv t^e \mod N$.

2. $\mathcal{A}^R$ runs $\mathcal{A}$ to obtain M^*.

3. $\mathcal{A}^R$ selects K and $K^* = K(\{m^*\})$. It computes $\hat{H} = iO(F\hat{D}H)$ (using z^* as the return value for the case when $m = m^*$).

4. $\mathcal{A}^R$ gives N, e, and $\hat{H}$ to $\mathcal{A}$.

5. $\mathcal{A}^R$ answers $\mathcal{A}$'s signature queries on $M \neq M^*$ by computing $F(K, h(M))^e \mod N$.

6. $\mathcal{A}$ outputs a signature σ^* on M^*, and $\mathcal{A}^R$ outputs σ^*.

Since $\hat{H}(M^*) = z^*$, and since σ^* is a valid signature on M^*, we have $(\sigma^*)^e \equiv z^* \mod N$ and hence $t = \sigma^*$. Thus $\mathcal{A}^R$ successfully computes e'th roots $\mod N$ with advantage $adv_2(\mathcal{A})$. Since the RSAP is assumed to be hard, the advantage of any such $\mathcal{A}^R$ must be negligible, and therefore, $adv_2(\mathcal{A})$ is also negligible.

But if $adv_2(\mathcal{A})$ is negligible, and $adv_2(\mathcal{A}) \approx adv_1(\mathcal{A}) \approx adv_0(\mathcal{A})$, then $adv_0(\mathcal{A})$ is also negligible. Hence the signature scheme is selectively secure. $\square$

Remark 4.3.5 (Random oracle model versus generic MLM model)**.** Theorem 4.3 is an interesting theoretical result. However, introducing iO in order to avoid the random oracle model obviously incurs a huge practical cost. Furthermore, in order to prefer the iO-based signature scheme to normal RSA-FDH from a security perspective, one would have to trust the random oracle model less than one trusts the assumptions that iO are built on. Ignoring the fact that random oracle based signature schemes have endured much more study than iO and multilinear map assumptions, proofs about iO schemes have thus far all been in generic multilinear map models of one form or another. For example, the Garg et al. NC^1 obfuscation algorithm is proven secure in the "generic coloured matrix" model, where roughly, they prove their scheme is secure as long as the attacker is restricted to attacks that involve multiplying the matrices of the branching program [31].

Ultimately, this scheme replaces the random oracle model with a different, less studied generic model at debilitating efficiency costs.

Chapter 5

Conclusions

Multilinear maps and indistinguishability obfuscation are powerful theoretical tools. They enable otherwise difficult cryptographic protocols such as one-round multi-party key exchange and existentially unforgeable signature schemes outside of the random oracle model.

5.1 State of the Art

There are essentially three multilinear map constructions: Graph-based multilinear maps [35], ideal lattice multilinear maps [29], and multilinear maps over the integers [22, 25].

1. Graph-based multilinear maps are very susceptible to zero-izing attacks. In the graph-based multilinear map construction, any two encodings of zero at the same level S can be used to learn the plaintext of any other encodings that happens to be at the right level (namely, a level that admits multiplication with level-S encodings). This is in contrast to the zero-izing attacks on the other multilinear map constructions that require *many* low-level encodings of zero (or at least, many different sets of encodings whose product is a top-level encoding of zero). See Section 4.1 of [35] for more details.

2. The original multilinear maps over ideal lattices have always suffered from "weak discrete log" attacks that render the decision-linear and subgroup membership problems easy in the ideal lattice setting [29].

3. The CLT13 multilinear map over the integers [22] originally appeared to avoid these weak discrete log attacks. However, the zero-izing attacks of Cheon et al. have shown that CLT13 is totally broken when low-level encodings of zero are available. For some time after the Cheon et al. discovery there were no known zero-ization attacks against specific applications of CLT13 where low-level encodings of zero were unavailable. Specifically, the Garg et al. obfuscation candidate appeared to avoid these attacks. Recently however, both obfuscation based on branching programs, and direct obfuscation of circuits were shown to be vulnerable to extensions of the Cheon et al. attack in certain circumstances [21].

   There were two independent attempts to fix CLT13, both of which failed to do so. However, multilinear maps over the integers have been tentatively restored by CLT15 [25]. CLT15 appears to avoid all zero-ization attacks, including the newest ones from [21]. There are currently no serious known attacks on CLT15. It is currently the only construction that is not vulnerable to some form of zero-ization.

   If κ is the multilinearity required, and λ is the security parameter, encodings in CLT15 have size $\mathcal{O}(\kappa^2\lambda^3)$. Zero-testing and multiplication are both equivalent in complexity to multiplying two $\mathcal{O}(\kappa^2\lambda^3)$ bit integers (when using the CLT15 efficiency heuristics). CLT15 is the most efficient construction available.

There are several viable obfuscation constructions that all give varying degrees of obfuscation. Currently, Zimmerman's direct obfuscation of circuits is the most efficient construction [45]. Circuits of size s, depth d, and input size n require multilinearity $\mathcal{O}(2^d n + n^2)$, but the obfuscation consists of only $\mathcal{O}(d^2s^2 + n^2)$ encodings, and the evaluation takes only $\mathcal{O}(d^2s^2 + n^2)$ multiplications of encodings. This is in contrast to the original obfuscation method using Barrington's Theorem that required multilinearity, obfuscation size, and evaluation time of $\mathcal{O}(4^d n + n^2)$.

Zimmerman's construction also achieves VBB with a "generic multilinear map" assumption, which is thus far the strongest form of security that has been proven for any obfuscation construction. It was shown recently that Zimmerman's construction fails to obfuscate certain simple circuits (such as point functions) when instantiated with CLT13 [21]. This attack illustrates a way in which the underlying GES can fail to satisfy "generic multilinear map" assumptions. Direct obfuscation remains intact when instantiated with CLT15.

Also noteworthy is the Koppula, Bishop, Waters (KBW) candidate for Turing Machine iO [40]. The purpose of this construction is to avoid the overhead involved in converting a program to a circuit. Indeed, circuits are not the most natural way to express programs.

Since circuits are acyclic graphs, there is no way to compactly express common programming constructs such as loops. In particular, the KBW construction has an obfuscation that grows with the Turing Machine size, rather than with the worst case running time (worst case running time is equivalent to circuit depth). However, this construction relies heavily on a circuit obfuscator. In particular, it requires obfuscating the encryption algorithm of some public key encryption scheme. This task is not currently feasible, so for the time being, iO for Turing Machines remains difficult.

5.2 Future Directions

There are two major problems with obfuscation and multilinear maps. First, they are both hopelessly impractical, with very large constants severely impacting size and speed. Second, they are both founded on new security assumptions which have undergone minimal scrutiny.

The impracticality of multilinear maps stems from the noise necessary for current constructions. Since the noise grows with each multiplication, and there is a limit on how much noise an encoding can have before it is no longer zero-testable, parameter sizes must increase as the number of multiplications desired increases. Most interesting applications of multilinear maps (obfuscation for example) require many multiplications, and therefore very large encodings. Improving multilinear maps to the point of usability will require reducing the size relative to the noise. Ideally, there should be no noise at all, and the size of each encoding should be independent of the number of multiplications necessary. An ideal multilinear map would enable realistic implementations of everything from NIKE to iO.

Obfuscation has an additional efficiency problem: the multilinearity required depends exponentially on the depth of the input circuit. Whereas it is possible to imagine using multilinear maps for very small multilinearity, obfuscation is not even remotely practical. Obfuscating a meaningful program necessarily requires a high degree of multilinearity. Reducing the dependency of encoding size on multilinearity would ameliorate this issue. Absent this, we would like to find an obfuscation algorithm whose multilinearity depends only polynomially on the depth of its input circuit.

Another direction to consider is the obfuscation of specific circuit families. While general-purpose obfuscation for all circuits is completely impractical, it may be easier to obfuscate specific circuits. There are many such circuit families we already know how to obfuscate efficiently (point-functions for example).

Very little has been proven about multilinear maps and obfuscation, and there have been only a few cryptanalytic attempts. The devastating zero-ization attacks on CLT13 certainly do not inspire confidence in multilinear maps as a whole. While CLT15 remains intact, much more cryptanalysis is necessary before we can be confident that the various security assumptions hold in the CLT15 setting. In this direction, Badrinarayanan et al. [2] have recently designed obfuscators for "evasive" functions (functions for which it is difficult to find a single input that maps to 0) that, in a certain generic multilinear model, provably does not allow encodings of 0 to be constructed at any level.

Security of obfuscation has thus far been proven only in generic multilinear map models that assume the security of the underlying multilinear map. These generic models mostly do not account for the ability of an adversary to perform zero-ization attacks. Furthermore, a number of researchers have proven VBB security in these generic multilinear map models. Given that VBB is impossible in the standard model, the usefulness of these generic models is questionable. The next step for obfuscation security is a model that manages to take into account known attacks on multilinear maps, while also being less generic so as to sidestep the VBB discrepancy.

Obfuscation and multilinear maps are very powerful new cryptographic tools, and making practical versions of them would be a huge breakthrough. The construction of an ideal multilinear map would mostly eliminate efficiency concerns. The construction of such a map is therefore the central open problem related to cryptographic multilinear maps.

References

[1] Prabhanjan Ananth, Divya Gupta, Yuval Ishai, and Amit Sahai. Optimizing obfuscation: Avoiding Barrington's theorem. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 646–658, New York, NY, USA, 2014. ACM.

[2] Saikrishna Badrinarayanan, Eric Miles, Amit Sahai, and Mark Zhandry. Post-zeroizing obfuscation: The case of evasive circuits. Cryptology ePrint Archive, Report 2015/167, 2015. http://eprint.iacr.org/.

[3] Boaz Barak, Oded Goldreich, Rusell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *Advances in Cryptology — CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 1–18. Springer Berlin Heidelberg, 2001.

[4] D A Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC1. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, STOC '86, pages 1–5, New York, NY, USA, 1986. ACM.

[5] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, CCS '93, pages 62–73, New York, NY, USA, 1993. ACM.

[6] Daniel J. Bernstein, Andreas Hülsing, Tanja Lange, and Ruben Niederhagen. Bad directions in cryptographic hash functions. In Ernest Foo and Douglas Stebila, editors, *Information Security and Privacy*, volume 9144 of *Lecture Notes in Computer Science*, pages 488–508. Springer International Publishing, 2015.

[7] Dan Boneh and Matt Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *Advances in Cryptology — CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. Springer Berlin Heidelberg, 2001.

[8] Dan Boneh, Craig Gentry, and Brent Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 258–275. Springer Berlin Heidelberg, 2005.

[9] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *Advances in Cryptology — ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532. Springer Berlin Heidelberg, 2001.

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

[11] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology — ASIACRYPT 2013*, volume 8270 of *Lecture Notes in Computer Science*, pages 280–300. Springer Berlin Heidelberg, 2013.

[12] Dan Boneh, Brent Waters, and Mark Zhandry. Low overhead broadcast encryption from multilinear maps. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014*, volume 8616 of *Lecture Notes in Computer Science*, pages 206–223. Springer Berlin Heidelberg, 2014.

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

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

[15] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014*, volume 8616 of *Lecture Notes in Computer Science*, pages 480–499. Springer Berlin Heidelberg, 2014.

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

[17] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, pages 309–325, New York, NY, USA, 2012. ACM.

[18] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, July 2004.

[19] Jung Hee Cheon, Kyoohyung Han, Changmin Lee, Hansol Ryu, and Damien Stehlé. Cryptanalysis of the multilinear map over the integers. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology — EUROCRYPT 2015*, volume 9056 of *Lecture Notes in Computer Science*, pages 3–12. Springer Berlin Heidelberg, 2015.

[20] Stephen A. Cook and H. James Hoover. A depth-universal circuit. *SIAM Journal on Computing*, 14(4):833–839, 1985.

[21] Jean-Sébastien Coron, Craig Gentry, Shai Halevi, Tancrède Lepoint, Hemanta K. Maji, Eric Miles, Mariana Raykova, Amit Sahai, and Mehdi Tibouchi. Zeroizing without low-level zeroes: New MMAP attacks and their limitations. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology — CRYPTO 2015*, volume 9215 of *Lecture Notes in Computer Science*, pages 247–266. Springer Berlin Heidelberg, 2015.

[22] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Practical multilinear maps over the integers. Cryptology ePrint Archive, Report 2013/183, 2013.

[23] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Practical multilinear maps over the integers. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology — CRYPTO 2013*, volume 8042 of *Lecture Notes in Computer Science*, pages 476–493. Springer Berlin Heidelberg, 2013.

[24] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Cryptanalysis of two candidate fixes of multilinear maps over the integers. Cryptology ePrint Archive, Report 2014/975, 2014. http://eprint.iacr.org/.

[25] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. New multilinear maps over the integers. Cryptology ePrint Archive, Report 2015/162, 2015.

[26] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. New multilinear maps over the integers. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology — CRYPTO 2015*, volume 9215 of *Lecture Notes in Computer Science*, pages 267–286. Springer Berlin Heidelberg, 2015.

[27] Jonathan Katz Alex J. Malozemoff Daniel Apon, Yan Huang. Implementing cryptographic program obfuscation. Cryptology ePrint Archive, Report 2014/779, 2014. http://eprint.iacr.org/.

[28] Eduarda S.V. Freire, Dennis Hofheinz, Kenneth G. Paterson, and Christoph Striecks. Programmable hash functions in the multilinear setting. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology — CRYPTO 2013*, volume 8042 of *Lecture Notes in Computer Science*, pages 513–530. Springer Berlin Heidelberg, 2013.

[29] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. Cryptology ePrint Archive, Report 2012/610, 2012.

[30] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology — EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 1–17. Springer Berlin Heidelberg, 2013.

[31] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. Cryptology ePrint Archive, Report 2013/451, 2013.

[32] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 40–49, Oct 2013.

[33] Sanjam Garg, Craig Gentry, Shai Halevi, Amit Sahai, and Brent Waters. Attribute-based encryption for circuits from multilinear maps. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology — CRYPTO 2013*, volume 8043 of *Lecture Notes in Computer Science*, pages 479–499. Springer Berlin Heidelberg, 2013.

[34] Sanjam Garg, Craig Gentry, Shai Halevi, and Mark Zhandry. Fully secure functional encryption without obfuscation. Cryptology ePrint Archive, Report 2014/666, 2014. http://eprint.iacr.org/.

[35] Craig Gentry, Sergey Gorbunov, and Shai Halevi. Graph-induced multilinear maps from lattices. In Yevgeniy Dodis and JesperBuus Nielsen, editors, *Theory of Cryptography*, volume 9015 of *Lecture Notes in Computer Science*, pages 498–527. Springer Berlin Heidelberg, 2015.

[36] Craig Gentry, Shai Halevi, Hemanta K. Maji, and Amit Sahai. Zeroizing without zeroes: Cryptanalyzing multilinear maps without encodings of zero. Cryptology ePrint Archive, Report 2014/929, 2014.

[37] Susan Hohenberger, Amit Sahai, and Brent Waters. Replacing a random oracle: Full domain hash from indistinguishability obfuscation. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology — EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 201–220. Springer Berlin Heidelberg, 2014.

[38] Antoine Joux. A one round protocol for tripartite Diffie-Hellman. In *Proceedings of the 4th International Symposium on Algorithmic Number Theory*, ANTS-IV, pages 385–394, London, UK, UK, 2000. Springer-Verlag.

[39] Joe Kilian. Founding crytpography on oblivious transfer. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 20–31, New York, NY, USA, 1988. ACM.

[40] Venkata Koppula, Allison Bishop Lewko, and Brent Waters. Indistinguishability obfuscation for turing machines with unbounded memory. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, STOC '15, pages 419–428, New York, NY, USA, 2015. ACM.

[41] Hyung Tae Lee and Jae Hong Seo. Security analysis of multilinear maps over the integers. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology — CRYPTO 2014*, volume 8616 of *Lecture Notes in Computer Science*, pages 224–240. Springer Berlin Heidelberg, 2014.

[42] Hyung Tae Lee and Jae Hong Seo. Security analysis of multilinear maps over the integers. Cryptology ePrint Archive, Report 2014/574, 2014.

[43] Franco P. Preparata and D.E. Muller. Efficient parallel evaluation of boolean expressions. *IEEE Transactions on Computers*, C-25(5):548–549, May 1976.

[44] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In Henri Gilbert, editor, *Advances in Cryptology*

— *EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 24–43. Springer Berlin Heidelberg, 2010.

[45] Joe Zimmerman. How to obfuscate programs directly. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology — EUROCRYPT 2015*, volume 9057 of *Lecture Notes in Computer Science*, pages 439–467. Springer Berlin Heidelberg, 2015.