Multi-Input Functional Encryption and Obfuscation

A Thesis
Presented to
The Established Interdisciplinary Committee for Mathematics and Natural Sciences
Reed College

In Partial Fulfillment
of the Requirements for the Degree
Bachelor of Arts

Sage R. Michaels

May 2018

Approved for the Division
(Mathematics)

Dylan McNamee

Acknowledgements

I want to thank a few people.

Abstract

This is an example of a thesis setup to use the reed thesis document class.

Table of Contents

Introduction

Chapter 1

Background

1.1 Encryption

Encryption is a way to share a message so that only the intended recipient(s) of that message are able to read it. Historically this was done by means of obscurity, in the sense that correspondents assumed only they knew the specific method by which messages between them would be encrypted. The problem with encryption via obscurity is that as soon as a method of encryption becomes popular, it also becomes obsolete.

1.1.1 Classical Encryption

Now, cryptographers work to develop encryption schemes that are computationally infeasible for adversaries to break even if the method of encryption is known (this is known as Kerckhoff's Principle). To do this, encryption functions are made public but take an extra parameter that is kept secret, we call this secret a key. The best keys are ones that are decently long and chosen randomly over some distribution (often uniformly), so it is infeasible for an adversary to check every possible key or guess the right key using information about communicating parties.

In defining an encryption scheme we call the set of all valid keys K called a key space, a set of all valid messages M called the message space, and a set of all valid cipher texts (encrypted messages) C called a cipher space.

Definition 1 (Classical Encryption Scheme).

$$Gen : \mathbb{Z} \to K \times K$$

Defined to be for $\lambda \in \mathbb{Z}$, $Gen(\lambda) \to (pk, sk)$ where pk and sk are random keys of length λ.

$$Enc : K \times M \to C$$

Defined for key $pk \in K$ and message $m \in M$; to be $Enc_{pk}(m) \to c$ for some cipher text $c \in C$

$$Dec : K \times C \to M$$

Defined for key $sk \in K$ and cipher text $c \in C$; to be $Dec_{sk}(c) \to m$ for some message $m \in M$

If $sk = pk$ this is called a symmetric or private key encryption scheme meaning only the correspondents know the key and they keep it secret. If $sk \neq pk$ then this is called an asymmetric or public key encryption scheme where sk is a secret key and pk is a public key. In a public key encryption scheme anyone can encrypt a message since the public key is public, but only people with the secret key are able to decrypt.

Definition 2 (Correctness)**.** *In this setting we say an encryption scheme Π is **correct** if for $n \in \mathbb{Z}, (sk, pk) \leftarrow Gen(n)$ and $m \in M$*

$$Dec_{sk}(Enc_{pk}(m)) = m$$

Suppose Alice and wants to send Bob a secret message m. To do this in the public key setting Bob would have to run Gen$(n) \to (pk, sk)$ and then send pk to Alice. Then Alice gets c :=Enc$_{pk}(m)$ and sends c over to Bob. Finally Bob gets m' :=Dec$_{sk}(c)$. If the scheme is correct then $m' = m$ and Bob is able to read Alice's message. The above interaction is represented in the following diagram.

Bob		Alice
Gen$(\lambda) \to (sk, pk)$		
	$\xrightarrow{pk}$	
		Enc$_{pk}(m) \to c$
	$\xleftarrow{c}$	
m' := Dec$_{pk}(c)$		

Table 1.1: Public Key Encryption Protocol

1.1.2 Functional Encryption

With classical encryption, decryption is all or nothing, either you have the secret key and can find out the message, or you don't have the secret key so you can't. With functional encryption we broaden the possibilities of what is communicated between senders in an encryption scheme. We start with a definition and then show the formal construction.

Definition 3 (Correctness)**.** *A Functional Encryption Scheme Π is **correct** if for $m \in M$, some predetermined function f with M as its domain, and appropriate public key and evaluation key $(pk, ek) \in K$ from Π's key generation algorithm:*

$$Dec_{ek}(Enc_{pk}(m)) = f(m)$$

It's easy to see that this definition encapsulates the older definition of correctness by making f the identity function $f(m) = m$, but this syntax covers many other cryptographic primitives as well like attribute based encryption (ABE) and identity based encryption (IBE). To see how these primitives are sub cases of functional encryption lets formalize our idea of a functional encryption scheme.

Definition 4 (Functional Encryption Scheme). *Given a message space M, a cipher space C, a family of functions $\mathcal{F}$, and a key space K , a functional encryption scheme Π is defined to be the following four algorithms:*

$$Setup : \mathbb{Z} \to K \times K$$

Defined for $\lambda \in \mathbb{Z}$; to be setup(λ) $\to$ (PP, MSK), generates a public parameter and master secret key both in K

$$Gen : K \times \mathcal{F} \to K$$

Defined for $f \in \mathcal{F}, MSK \in K$; to be Gen$(mk, c) \to SK_f$ which is kept secret and is the secret key associated with the functionality of f.

$$Enc : K \times M \to C$$

Defined: For $PP \in K$ and $m \in M$; to be $Enc_{PP}(m) \to c$

$$Dec : K \times C \to M$$

Defined for $SK_f \in K$ and $c \in C$; to be Dec$(SK_f, c) \to m'$ if the scheme is correct then $m' = f(m)$ for some previously encrypted m and some $f \in \mathcal{F}$.

Suppose Alice wants to store some text files ($\{m_1, m_2, \cdots, m_n\}$) on Bob's Cloud Storage to retrieve at some later date. Bob, owner of Bob's Cloud Storage, has a strong dislike for cats and doesn't want anything stored on his cloud with the word "cat" in it. Alice and Bob contact Janet, an impartial third party, and share their desired functionalities: Alice wants to be able to fully decrypt (i.e. identity functionality), Bob wants to be able to check if the word "cat" is in anything stored on his server (we call this function checking for the word "cat" f). Janet runs Setup(λ) $\to$ (PP, MSK) and publishes the public parameter, and also runs Keygen$(MSK, id) \to SK_{id}$ and Keygen$(mk, f) \to SK_f$. Then Janet securely sends sk_{id} to Alice and sk_f to Bob. Now Alice encrypts her text files Enc$_{pk}(m_1) \to c_1$,Enc$_{pk}(m_2) \to c_2, \cdots$,Enc$_{pk}(m_n) \to c_n$, and sends $\{c_1, c_2, \cdots, c_n\}$ to Bob for storage. Bob then runs Dec$_{sk_f}(c_1) \to b_1, \cdots,$ Dec$_{sk_f}(c_n) \to b_n$ and allows storage of all c_i that indicate the word "cat" is not being stored. This interaction is illustrated in the diagram below:

$$\begin{tikzcd}
\text{Bob} & \text{Janet} & \text{Alice} \\
{} & \mathrm{Setup}(\lambda) \to (MSK, PP) \arrow[l, "PP"'] \arrow[r, "PP"] & {} \\
{} \arrow[r, "f"] & {} & {} \arrow[l, "\mathrm{id}"'] \\
& \mathrm{KeyGen}(id, mk) \to SK_{\mathrm{id}} \arrow[r, "\mathrm{secure}(SK_{\mathrm{id}})"] & {} \\
{} & \mathrm{KeyGen}(f, mk) \to SK_f \arrow[l, "\mathrm{secure}(SK_f)"'] & \\
& & \mathrm{Enc}_{PP}(m_1) \to c_1 \\
& & \vdots \\
& & \mathrm{Enc}_{PP}(m_n) \to c_n \\
{} & & \{c_1, \cdots, c_n\} \arrow[ll] \\
\mathrm{Dec}_{SK_f}(c_1) \to b_1 & & \\
\vdots & & \\
\mathrm{Dec}_{SK_f}(c_n) \to b_n & &
\end{tikzcd}$$

It should be noted that the impartial 3rd party does not need to be fully trusted since Janet doesn't have any of Alice's cipher texts to decrypt, but it is important that Janet is impartial since otherwise should could collaborate with Alice to store a Cat Encyclopedia on Bob's Servers, or could collaborate with Bob to learn more about what Alice is storing than what Alice and Bob have agreed upon.

Research into functional encryption is currently at early stages, later we will describe the details of how it works, and where it's currently limited. Even at such an early stage cryptographic research, functional encryption has proven to be a powerful tool giving us a variety of functionaries with only small alterations of syntax such as attribute based encryption, identity based encryption , and multi-party input functional encryption [Boneh et al. (2011)] which we will actually work with later on.

1.2 Obfuscation

1.2.1 Black Box Obfuscation

Circuit (and program) obfuscation is a cryptographic method that seeks to make a circuit unintelligible to anyone looking at only its obfuscation. Black box obfuscation as defined in [Barak et al. (2001)] the goal of black box obfuscation is that a circuit that has been black box obfuscated should reveal no more about it's inner workings than a table of inputs and outputs for the same circuit (we call this table an Oracle).

Definition 5. *Given any circuit C and an Oracle $\mathcal{O}_c(\cdot)$ with the same functionality of C, an obfuscater $\mathcal{O}(\cdot)$, and any adversary $\mathcal{A}$. We say $\mathcal{O}$ is a Black Box obfuscater if $\mathcal{A}$ given access to $\mathcal{O}(C)(\cdot)$ can determine just as much about the inner workings of C as if $\mathcal{A}$ had been given access to $\mathcal{O}_c(\cdot)$.*

Let's take a moment to talk about why cryptographers would want black box obfuscation as a tool. Currently public key encryption relies on expensive computations, (for example in RSA the private keys are just inverses of public keys in some group large enough for this to be difficult to compute). However, private key encryption is much more efficient since it's just running some sort of permutation on the message using the secret key and then descrambling the cipher text with the same secret key. Using black box obfuscation if would be possible to obfuscate a private key encryption function with a secret key sk baked in $\mathcal{O}(\mathrm{Enc}_{sk}(\cdot) \rightarrow \mathrm{Enc}(\cdot)$ where $\mathrm{Enc}(\cdot)$ can be made public without risk of anyone learning sk keeping the ability to decrypt in the hands of those who started off with the secret key. Thus Black Box Obfuscation would make efficient private key encryption into efficient public key encryption.

1.2.2 Indistinguishability Obfuscation

While Black Box Obfuscation was proved impossible in [Barak et al. (2001)], cryptographers weakened their definition of obfuscation to try and see what *is* possible in the field of obfuscation. This led to a new notion of obfuscation called Indistinguishability Obfuscation.

Before defining indistinguishability obfuscation it is worth noting that a distinguisher is any polynomial time algorithm that can differentiate two things. For example, a distinguisher for apples and oranges might look at the color of what it has been given and output Orange if the color is orange or Apple if the color of what it's been given is red/green/yellow, however another distinguisher for apples and oranges might just flip a coin and output Apple if the coin comes up heads and Orange otherwise, the second distinguisher would probably not be a very good distinguisher. Distinguishers are only defined on the two things that they distinguish, apples or oranges, green or blue, paper or not paper.

Definition 6. *Indistinguishability Obfuscation*

Given circuits C_0, C_1 where $|C_0| \approx |C_1|, C_0(x) = C_1(x)$ for all valid x, and an obfuscater $i\mathcal{O}(\cdot)$; we say that $i\mathcal{O}$ is an ***indistinguishability obfuscater*** *if for all Distinguishers $\mathcal{D}$*

$$\Pr[\mathcal{D}(i\mathcal{O}(C_0)) = 1] - \Pr[\mathcal{D}(i\mathcal{O}(C_1)) = 1] \leq negl$$

This definition can be a little confusing to understand. In short, Indistinguishability Obfuscation guarantees that two circuits with the same functionality are indistinguishable from one another once run through an indistinguishability obfuscater.

What can be more confusing is why this would be useful since we require the two circuits to be functionally the same. The most simple usage is in removing water marks from programs. Suppose Dan buys a fancy piece of software called Macrosoft Word for his company. Macrosoft is worried about their software being pirated so they put a watermark in Dan's copy of Macrosoft Word that doesn't change the functionality of his copy of the program, indicating that this is Dan's copy. Suppose Dan wants to make Macrosoft Word free for everyone and decides to post it on a torrenting website. If Dan posts his copy as is, Macrosoft and their copyright lawyers will be able to see that it was Dan who illegally shared his copy of their software. Instead, if Dan first runs his copy of Macrosoft word through an Indistinguishability Obfuscater and post that, Macrosoft and their lawyers will be unable to tell if it was Dan's copy that was posted, or another customer's.

This might seem like a weak definition, but indistinguishability obfuscation has become a powerful cryptographic tool, and has proven to be the best possible obfuscation, the proof is short and eloquent so we will show it here but it was originally formulated by [Goldwasser & Rothblum (2007)].

Proof. Let $i\mathcal{O}(\cdot)$ be an indistinguishability obfuscater, $\mathcal{O}(\cdot)$ the best obfuscater possible, and C a circuit. Then by definition of Indistinguishability Obfuscation, $i\mathcal{O}(C)$ is indistinguishable from $i\mathcal{O}(\mathcal{O}(C))$. Thus Indistinguishability Obfuscation is at least as good as any other type of Obfuscation. □

Because of this, Cryptographers often treat Indistinguishability Obfuscation the same as Black Box Obfuscation. While this could lead to issues in the future and there is lots of room for work to be done in quantifying degrees of Obfuscation, we will fall back on the assumption that the best possible obfuscation is good enough for our purposes.

1.3 Bilinear Maps

Lastly we give a brief introduction to Bilinear maps, which we will later generalize to Multilinear maps.

1.3.1 Definition

Definition 7. *Bilinear Maps*

Let G_1, G_2, G_3 be cyclic groups of prime order p. Then we say a map $e : G_1 \times G_2 \rightarrow G_3$ is bilinear if

1. *For all $g_1 \in G_1, g_2 \in G_2$, and $\alpha \in \mathbb{Z}_p$, $e(\alpha \cdot g_1, g_2) = e(g_1, \alpha \cdot g_2) = \alpha \cdot e(g_1, g_2)$*

2. *For generators $g_1 \in G_1$, $g_2 \in G_2$, and $g_3 \in G_3$, $e(g_1, g_2) = g_3$*

1.3.2 Intuition

The most accessible example of Bilinear Map is in Tripartite Diffie-Hellman Key Exchange. Here the Bilinear Map $e : G_1 \times G_1 \rightarrow G_2$ is defined $e(g_1^a, g_1^b) \rightarrow g_2^{a \cdot b}$ for generators $g_1 \in G_1$ and $g_2 \in G_2$. We can think of a, b as secrets and g_1^a, g_1^b as their encodings in g_1. Using the map e we can multiply secrets and by multiplying encodings $g^a \cdot g^b = g^{a+b}$ we can also add secrets.

Suppose Alice, Bob and Carol wish to agree upon a single secret key for the three of them. To do so Alice would run an instance generation function that outputs a description of the group G_1, a generator g_1 and a bilinear map e as we defined above. Then she would choose a secret $a \in \mathbb{Z}_p$ and broadcast $(e \in \textbf{params}, g_1, pk_a := g_1^a)$. Then Bob would choose $b \in \mathbb{Z}_p$ and broadcast $pk_b := g_1^b$ and Alice would do the same broadcasting $pk_c := g_1^c$. For each party to learn the secret key they would need to run:

$$\begin{aligned} \text{Alice: } & e(pk_b, pk_c)^a = e(g_1^b, g_1^c)^a = (g_2^{b \cdot c})^a = g_2^{a \cdot b \cdot c} = sk \\ \text{Bob: } & e(pk_a, pk_c)^b = g_2^{a \cdot b \cdot c} = sk \\ \text{Carol: } & e(pk_a, pk_b)^c = g_2^{a \cdot b \cdot c} = sk \end{aligned}$$

The important thing to understand is that this bilinear map makes it possible for 3 parties to share a secret without risk of an eavesdropper also being able to know the secret. To generalize, a k-multilinear map allows us to encode k secrets easily, so if each party involved in the exchange knows a secret this allows for $k + 1$ non interactive key exchange.

1.3.3 Hardness Assumption

In order to talk about how secure it is to run protocols that use bilinear maps (such as Alice, Bob, and Carol's key exchange) we need to formalize what the adversary would need to be able to do in order to compromise security. We call these formalizations *hardness assumptions.*

First we define the following problems.

Definition 8. *Bilinear Computational Diffie-Hellman Problem*

Let G_1, G_2 be cyclic groups of prime order p and $e : G_1 \times G_1 \rightarrow G_2$ be a bilinear map. For $a, b, c, \in \mathbb{Z}_p$ chosen uniformly and g_1, g_2 generators of G_1, G_2 respectively. The Computational Diffie-Hellman problem is: given e, g_1, g_1^a, g_1^b and $,g_1^c$; compute

$$e(g_1, g_1^{a \cdot b \cdot c}) = g_2^{a \cdot b \cdot c}$$

The computational Diffie-Hellman problem might seem a bit confusing, but it is pretty much just a formalization of what we realized after discussing tripartite key exchange: multiplying two secrets is easy using bilinear maps, but that multiplying by a third secret is difficult. Thus we have

Hardness Assumption 1. *Bilinear CDD hardness assumption*
The bilinear computational Diffie-Hellman problem is hard.

Another hardness assumption that we need to make is the bilinear discrete logarithm problem defined below.

Definition 9. *Bilinear Discrete Logarithm Let G_1, G_2 be cyclic groups of prime order p and $e : G_1 \times G_1 \to G_2$ be a bilinear map. For $a \in \mathbb{Z}_p$ chosen uniformly, and generators g_1, g_2 of G_1, G_2 respectively, the bilinear discrete logarithm problem is: given $g_1, g_1^\alpha, G_1, G_2$, and e find α.*

Note that while the above problem may seem easy, the encoding $g_1^\alpha = g$ just looks like some element of G_1, and since g_1 is a generator of G_1, *any* element of G_1 can be written as $g_1^\beta\ \forall \beta \in \mathbb{Z}_p$. We want it to be difficult to get a secret out of its encoding, so we also rely on the following assumption.

Hardness Assumption 2. *Bilinear DL hardness assumption*
The bilinear discrete logarithm problem is hard

There are arguments about if these are *good* assumptions to be making, but these arguments are outside the scope of this thesis. It is enough for us that so far cryptographers and mathematicians have been unable to give efficient solutions to solving these problems in general cases.

Chapter 2

Multi-Linear Maps

2.1 Cryptographic Multilinear Maps

2.1.1 Dream Definition

In order to construct indistinguishability obfuscation or functional encryption, we need to be able to do computations in public on encoded secrets in such a way that only information about the final output can be discerned. Computationally, we want to be able to encode, add, multiply and negate secrets.

In the previous chapter, we saw that bilinear maps allow us to multiply two encoded secrets securely, but if we want to be able to do more complex computations involving multiple multiplications, or sums of products, or products of sums, we need a new cryptographic tool that allows for greater functionality.

Definition 10. *Cryptographic Multilinear Map*

Let $G_1, G_2, \cdots, G_k, G_T$ be cyclic groups each of the same prime order p with generators $g_1, g_2, \cdots g_k$ respectively.. Then we say a map $e : G_1 \times G_2 \times \cdots \times G_k \to G_T$ is a k-multilinear map if

1. *For $g_i \in G_i \; \forall i \in [K]$ where $[K]$ is the set $\{1, 2, \cdots, k\}$, and $\alpha \in \mathbb{Z}_p$ then $e(g_1, g_2, \cdots, g_\ell^\alpha, \cdots, g_k) = e(g_1, g_2, \cdots, g_\ell, \cdots, g_k)^\alpha$*

2. *if g_i generates G_i then $e(g_1, g_2, \cdots, g_k)$ generates G_T. We say that e is **non-degenerate** if this property holds.*

For $g_i \in G_i$ and $\alpha \in \mathbb{Z}_p$, we can think of g_i^α as a public encoding of secret α. In this setting, we would use the map e to do k multiplications of encodings in our secure computation. For example for $a, b, c, d, e, f, h \in \mathbb{Z}_p$ and generators $g_1 \in G_1, g_2 \in G_2, g_3 \in G_3$ and $g_4 \in G_4$we could compute $(a + b) \cdot (c + d) \cdot (e + f + h)$ with the 3-linear map $e : G_1 \times G_2 \times G_3 \to G_4$ with

$$\begin{aligned} e(g_1^a \cdot g_1^b, g_2^c \cdot g_2^d, g_3^e \cdot g_3^f \cdot g_3^h) &= e(g_1^{a+b}, g_2^{c+d}, g_3^{e+f+h}) \\ &= e(g_1, g_2, g_3)^{(a+b)(c+d)(e+f+h)} \\ &= g_4^{(a+b)(c+d)(e+f+h)} \end{aligned}$$

2.1.2 Hardness Assumption

As before we would like to formalize what is being assumed to be difficult in order to be able to discuss the security of using multilinear maps to do computations on secrets in the clear. They are just an extensions of the bilinear discrete logarithm problem and decisional Diffie-Hellman problem.

Definition 11. *Multilinear Discrete-Log*

The same as for the Bilinear DL problem except an adversary is also given the index $i < k$ to know which group they are working in.

Definition 12. *Multilinear Decisional Diffie-Hellman Problem*

Let $\alpha, \alpha_1, \alpha_2, \cdots \alpha_k, \alpha_{k+1} \in \mathbb{Z}_p$ be chosen uniformly. The MDDH problem is: given $G_1, G_2, \cdots, G_k, G_T$ cyclic groups each of the same prime order p with generators $g_1, g_2, \cdots, g_k$ respectively, $g_1^{\alpha_1}, g_2^{\alpha_2}, \cdots, g_i^{\alpha_i}, g_i^{\alpha_{i+1}}, \cdots g_k^{\alpha_{k+1}}$ and multilinear map e; distinguish

$$\prod_{j=1}^{k+1} \alpha_j \cdot e(g_1, g_2, \cdots, g_k) \quad \textit{from} \quad \alpha \cdot e(g_1, g_2, \cdots, g_k)$$

We assume both of these problems to be hard.

2.2 Graded Encoding Schemes

2.2.1 Intuition

To help build an intuition for how Graded Encoding Schemes work, we begin with a simplified definition, that allows us to use familiar syntax, and then we'll build up to the actual definition from there.

Definition 13. *k-Graded Encoding Scheme (simplified)*

Given cyclic groups $G_1, G_2, \cdots, G_k$ of prime order p. We call the family of bilinear maps

$$e_{i,j} : G_i \times G_j \to G_{i+j} \textit{ for all } 0 \le i, j \le k \textit{ where } i + j \le k$$

a simplified Graded Encoding Scheme. For any secret $\alpha \in \mathbb{Z}_p$ and generator g_i of G_i we call g_i^{α} a level i encoding of α where elements of $\mathbb{Z}_p$ are called level zero encodings.

In this Encoding Scheme, secrets $\alpha_1, \alpha_2, \cdots \in \mathbb{Z}_p$ are initially encoded as they were with bilinear maps, by taking a generator $g_1 \in G_i$ and raising it to the α_1 so that it looks like $g_1^{\alpha_1}$. As stated in the definition we call this a level one encoding.

For a level one encoding of elements, computation on secrets is the same as before, but now if we have $e_{1,1}(g_1^{\alpha_1}, g_1^{\alpha_2}) = g_2^{\alpha_1 \cdot \alpha_2}$ for generator g_2 of G_2 and want to continue to do computations on the new secret $\alpha_1 \cdot \alpha_2$ with another secret α_3 we can 'multiply' the level one encoding α_3 with a level one encoding of 1 to get $e_{1,1}(g_1^{\alpha_3}, g_1^{1}) = g_2^{\alpha_3}$ and continue to do operations as before such as $e_{2,2}(g_2^{\alpha_3}, g_2^{\alpha_1 \cdot \alpha_2}) = g_4^{\alpha_3 \cdot \alpha_1 \alpha_2}$ for g_4 generator of G_4. So using this scheme, we can do computations on multiple secrets as long as there are at most k multiplications.

2.2.2 Encodings

As we said in the beginning of this section, we gave a simplified definition of graded encoding schemes in order to give intuition into the *graded* structure of the scheme. In our simplified definition, encodings are unique and of the form g_i^α. In practice there are sets of randomized valid encodings for a secret $\alpha \in \mathbb{Z}_p$ at any level, we denote the set of valid encodings of α at level $i : S_i^{(\alpha)}$.

Definition 14. *Level i Encoding*

Let R be a cyclotomic ring, $R_q = R/qR$ for prime q, $g \in R_q$ be small, $z \in R_q$ chosen randomly (so it won't be small), and $I = \langle g \rangle$ the principal prime ideal of g. We define a valid level i encoding of $\alpha \in R_g \cong \mathbb{Z}_p$ to be anything in the set

$$S_i^\alpha = \left\{ \left[\frac{a}{z^i} \right]_q \middle| \, a \in \alpha + I \text{ and } a \text{ is small} \right\}$$

For more details into the definition of short, long, and why it is safe to assume z is invertible in this ring, see [Garg et al. (2013a)], we will instead show how this encoding allows us to achieve multilinear map like functionality.

Let $a \in S_0^\alpha$ and $b \in S_0^\beta$. Then

$$\left[\frac{a}{z^i}\right]_q \cdot \left[\frac{b}{z^j}\right]_q = \left[\frac{a \cdot b}{z^{i+j}}\right]_q$$

So multiplication of a level i encoding with a level j encoding gives a level $i + j$ encoding of their product.

$$\left[\frac{a}{z^i}\right]_q + \left[\frac{b}{z^i}\right]_q = \left[\frac{a + b}{z^i}\right]_q$$

Thus this encoding gives us the properties we were looking for in a graded encoding scheme.

2.2.3 Definition

Now we are ready for the formal definition of graded encoding schemes, after which we will formalize the procedures that we expect along with such a graded encoding scheme.

Definition 15. *k-Graded Encoding Scheme*

A k-Graded Encoding System consists of a ring R and a system of sets $\mathcal{S} = \{S_i^{(\alpha)} \subset \{0,1\}* \,|\alpha \in R,\ 0 \le i \le k\}$, such that:

1. *For every fixed index i, the sets $\{S_i^{(\alpha)} | \alpha \in R\}$ are disjoint (meaning they form a partition of $S_v := \cup_\alpha S_v^{(\alpha)}$)*
2. *There is an associative binary operation '+' and a self-inverse unary operation '−' such that $\forall \alpha_1, \alpha_2 \in R$, index $i \le k$, and $u_1 \in S_i^{(\alpha_1)}$ and $u_2 \in S_i^{(\alpha_2)}$, it holds that*

$$u_1 + u_2 \in S_i^{(\alpha_1+\alpha_2)} \text{ and } -u_1 \in S_i^{(-\alpha_1)}$$

where $\alpha_1 + \alpha_2$ and $-\alpha_1$ are addition and negation in R.

3. *There is an associative binary operation '$\times$' (on $\{0,1\}*$) such that for every $\alpha_1, \alpha_2 \in R$ every i_1, i_2 with $i_1 + i_2 \leq k$ and every $u_1 \in S_{i_1}^{(\alpha_1)}$ and $u_2 \in S_{i_2}^{(\alpha_2)}$ it holds that*

$$u_1 \times u_2 \in S_{i_1+i_2}^{(\alpha_1 \cdot \alpha_2)}$$

Where $\alpha_1 \cdot \alpha_2$ is multiplication in R, and $i_1 + i_2$ is integer addition.

2.2.4 Procedures

- **Instance Generation**:
  - Inputs: λ, our security parameter, and k the maximum number of multiplication our computation might need (note how this will end up being the degree of our k-graded encoding scheme)
  - Outputs: $\mathcal{S}$, the description of a k graded encoding scheme as described above, and p_{zt} a level k zero-test parameter which we will explain with some detail in the next section.
- **Ring Sampler**: takes $\mathcal{S}$ as input and outputs a level-zero encoding $a \in S_0^{(\alpha)}$ for uniform $\alpha \in R$. The details of how this procedure is constructed is important, but outside the scope of this thesis. What is important to understand is that because this gives an encoding of a uniform *plaintext* element of R without indicating what that plaintext element is, there is a negligible probability that an adversary would be able to get and recognize a plaintext encoding of 1 or something else that would compromise the security of the multilinear map. However this does give companions of the party that ran the Instance Generation the ability to participate in k-nary key exchange by running the Ring Sampler, saving their plaintext encoding, and then publicizing a higher level encoding.
- **Encoding**: takes as input $\mathcal{S}$, the level zero encoding $a \in S_0^{(\alpha)}$, and i and then outputs $u \in S_i^{(\alpha)}$, a level i encoding of a
- **Addition, negation, multiplication**: all as we've already described

2.2.5 Zero Test

It is important to note that while randomizing encodings makes the scheme harder to break in many ways, it also makes getting information from encoded values much more difficult. In fact it makes it so much more difficult to get information from encoded values that we can only check to see if it is or isn't zero. In other words, we need a way to determine wether or not $\left[\frac{u}{z^k}\right]_q$ is in $0 + I = I = \langle g \rangle$. We call this operation the zero test. To do this we need to define a new *zero test* parameter which we will denote p_{zt}.

Definition 16. *Zero Test*

Let R be a cyclotomic ring, $R_q = R/qR$ for prime q, $g, h \in R_q$ be small, $z \in R_q$ chosen randomly (so it won't be small), and $I = \langle g \rangle$ the principal prime ideal of g. Define the zero test parameter to be

$$p_{zt} = \left[\frac{h \cdot z^k}{g}\right]_q$$

Now for arbitrary level k encoding u_k of some unknown plaintext u we define the zero test to be 1 if $p_{zt} \cdot u_k$ is small and 0 if $p_{zt} \cdot u_k$ is large.

As before we won't go into the details of small and large, but the designation of small or large in the zero test sense is consistent with our earlier usages of those terms. Below is how we are able to link our earlier usages of the words small and large with the zero test.

$$\begin{aligned} p_{zt} \cdot u_k &= \left[\frac{h \cdot z^k}{g}\right]_q \cdot \left[\frac{u}{z^k}\right]_q \text{ since } u_k \in u + I \text{ defined to be small } r \text{ must be small} \\ &= \left[\frac{h \cdot (u + r \cdot g)}{g}\right]_q \end{aligned}$$

If u_k is an encoding of zero then

$$\begin{aligned} &= \left[\frac{h \cdot (0 + r \cdot g)}{g}\right]_q \\ &= \left[h \cdot r\right]_q \end{aligned}$$

where both h and r are small so $h \cdot r$ is small.

Otherwise if u_k is not an encoding of zero then

$$\left[\frac{h \cdot (u + r \cdot g)}{g}\right]_q \approx \left[\frac{h \cdot u}{g}\right]_q$$

Where h and g are small and u is probably not small so

$$\left[\frac{h \cdot u}{g}\right]_q$$

is big.

2.2.6 Hardness Assumptions

The hardness assumptions we have discussed up to this point are pretty standard and well studied, but with the creation of new cryptographic tools comes new hardness assumptions. For this graded encoding scheme a new hardness assumption had to be formulated and while it is inspired by DDH.

Definition 17. *Graded DDH*

```
1. $(\mathcal{S}, p_{zt}) \leftarrow InstGen(\lambda, k)$
2. For $i = 1, \cdots, k+1$ :
3.     Choose $\alpha_i \leftarrow samp(\mathcal{S})$        # get $k+1$ level-0 encodings
4.     Set $u_i \leftarrow encode(\mathcal{S}, 1, \alpha_i)$    # get level-1 encodings of $\alpha_i$
5. Set $\tilde{\alpha} = \prod_{i=1}^{k+1} \alpha_i$                # product of $k+1$ level-0 encodings
6. Choose $\bar{a} \leftarrow samp(\mathcal{S})$              # get level-0 encoding of random element
7. Set $\tilde{u} \leftarrow encode(\mathcal{S}, k, \tilde{a})$        # level-k encoding of the product
8. Set $\bar{u} \leftarrow encode(\mathcal{S}, k, \bar{a})$        # level-k encoding of random element
9. $\boldsymbol{u} \leftarrow \{\tilde{u}, \bar{u}\}$ chosen uniformly
```

Given $\mathcal{S}, u_1, u_2, \cdots, u_{k+1}$, and $\boldsymbol{u}$, determine if $\boldsymbol{u}$ is $\tilde{u}$ or $\bar{u}$.

As with our other DDH definitions, when we assume this is hard, we are really saying that in a k-graded encoding scheme, it is hard to determine the product of $k+1$ level one encodings. We won't go much into how *good* this assumption is, we will say that this encoding scheme is far less studied than, say, RSA and so the cryptanalysis is much less extensive. However there has been some cryptanalysis done on graded encoding schemes [Garg et al. (2013a)] and overall, it seems that depending on what type of security is needed, different variations on the scheme can be used to protect against different attacks.

Chapter 3

Indistinguishability Obfuscation

3.1 Circuit Representation

While multilinear maps are a central tool used for the obfuscation of circuits, it is difficult to see how when most of us think of circuits as boolean expressions of the form

$$C(x, y, z) = (x \cdot y) + z$$

which denotes the circuit (x and y) or z. A first thought might be to encode x, y and z with a k-graded encoding scheme, however this would not hide the operations used in C, only the values of the starting parameters. For circuit obfuscation we want it to be hard to determine what C is.

3.1.1 Branching Programs

We want to find another way of representing a circuit C that will allow us to obscure its inner workings. To do this we work with variations of a data structure called a branching program. We start with an example of a branching program to help give an intuition, note that θ and I indicate the output of the program to be 0 and 1 respectively.

For inputs $x_1, x_2, x_3, x_4 \in \{0, 1\}$ a branching comparison program for $x_1x_2 > x_3x_4$ is given by:

i	inp(i)	if inp(i) $= 0$ go to	if inp(i) $= 1$ go to
1.	1	2	3
2.	3	4	θ
3.	3	I	4
4.	2	θ	5
5.	4	I	θ

In the above program, the first column i indicates the current line of the program we are at, the second column inp(i) indicates the index of the input to focus on (i.e. inp(1) indicates to focus on x_1). The third and fourth columns give us either the next step of the program to go to depending on the value of inp(i); or an output value θ or I.

With this intuition we formalize a variant of branching programs that uses matrices rather than line numbers to get the same functionality.

Definition 18. *(Oblivious) Matrix Branching Program*
An oblivious branching program of length-n for ℓ-bit inputs is a sequence

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

Where the $A_{i,b}$'s are permutation matrices in $\{0,1\}^{5\times 5}$, and $inp(i) \in [\ell]$ is the input bit position examined at step i of the branching program. The function computed by this branching program is

$$f_{BP,A_0,A_1}(x) := \begin{cases} 1 & \text{if } \prod_{i=1}^{n} A_{i,x_{inp(i)}} = I \\ 0 & \text{otherwise} \end{cases}$$

Where in the example the program chose one of two line numbers to jump to depending on the value of the bit at inp(i), in matrix branching programs one of two matrices is chosen: $A_{i,0}$ or $A_{i,1}$.

3.1.2 Barrington's Theorem

Originally branching programs were used for oblivious two party computation, but we can start to use them in the context of obfuscating circuits with the following theorem.

Theorem 1. *Barrington's Theorem*
For any depth d fan-in-2 circuit C, there exists a branching program consisting of at most 4^d permutation matrices $A_{i,b} \in \{0,1\}^{5\times 5}$ that computes the same function as the circuit C.

Note that in order to keep these branching programs efficient we need our circuit to have logarithmic depth, so this confines us to circuits in Nick's Class (NC^1) although later we will generalize to polynomial sized circuits.

3.2 Approaching Obfuscation for NC^1

Now that we understand how circuits can be represented as a product of matrices, we can start to outline how we use branching programs and graded encoding schemes to create circuit obfuscation for circuits in NC.

3.2.1 Randomized Branching Programs

An easy first step in obfuscating a branching program is to randomize it.

Definition 19. *Randomized Branching Programs*

Given a branching program $\mathcal{BP} = (inp(i), A_{i,0}, A_{i,1})_{i=1}^n$ *and* n *random* 5×5 *invertible matrices* $R_1, R_2, \cdots, R_n$*. A random branching program is the branching program made up of* $\tilde{A_{i,b}} = R_{i-1} \cdot A_{i,b} \cdot R_i^{-1}$ *of the form*

$$\mathcal{RBP} = (inp(i), \tilde{A_{i,0}}, \tilde{A_{i,1}})_{i=1}^n$$

Not only does randomization help to obscure the underlying circuit, it also prevents any sort of attack that involves reordering the matrices, since matrix multiplication isn't commutative.

3.2.2 Kilian's Protocol

The first attempt of making indistinguishability obfuscation out for randomized branching programs was Kilian's protocol described below.

Suppose Alice is the obfuscater and Bob is the evaluator. Alice wants to obfuscate circuit $C \in \mathcal{C}$ where $\mathcal{C}$ is the circuit family of all circuits with the same domain and codomain as C. Alice starts with a universal circuit $U(\cdot, \cdot)$ which has the property that given the encoding of a circuit $C \in \mathcal{C}$, $U(C, m) = C(m)$ for all valid messages m.

Alice then gets the randomized branching program of

$$U = \mathcal{RBP}_U = (\text{inp}(i), \tilde{A_{i,0}}, \tilde{A_{i,1}})_{i=1}^n$$

Then Alice selects all $\tilde{A}_{i,x_{\text{inp}(i)}}$ for all i with $\text{inp}(i)$ corresponding to a bit in the encoding of C. Once this is done Alice sends the partially selected random branching program to Bob who selects matrices corresponding to his input y.

It's worth thinking about why this is a good start for obfuscation. When Bob receives the branching program he has no way of knowing which of the two matrices Alice selected, all he sees of the matrices corresponding to her inputs are $\tilde{A}_i$, just the matrix itself and its order in the product. Also, the randomization prevents Bob from being able to reorder the matrices of the branching program

Unfortunately, Kilian's protocol alone is broken. However we can further strengthen this scheme using a graded encoding scheme.

3.2.3 Encoded Random Branching Programs

Our first potential fix for Kilian's protocol uses the tool described in chapter 2, graded encoding schemes. As before Alice has a random branching program of a universal circuit, she selects the matrices corresponding to her input (i.e. the encoding of C); but now she generates a n-graded encoding scheme $\mathcal{S}$ with maps $e_{i,j}$, (where $n = 4^d$ and d is the depth of her original circuit C) and replaces the entries of each matrix with there level-1 encoding. Now, when Bob multiplies the encoded random branching program together, Alice can send over a level k encoding of the identity matrix so that Bob can zero test to see if the output of his computation is 0 or 1 on his input.

For example, in Killian's Protocol $A_{i,\mathrm{inp}(i)}$ might be of the form

$$A_{i,\mathrm{inp}(i)} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

but in the encoded random branching programs construction

$$A_{i,\mathrm{inp}(i)} = \begin{bmatrix} a_{1,1} \leftarrow S_1^1 & a_{1,2} \leftarrow S_1^0 & a_{1,3} \leftarrow S_1^0 & a_{1,4} \leftarrow S_1^1 & a_{1,5} \leftarrow S_1^0 \\ a_{2,1} \leftarrow S_1^0 & a_{2,2} \leftarrow S_1^1 & a_{2,3} \leftarrow S_1^0 & a_{2,4} \leftarrow S_1^0 & a_{2,5} \leftarrow S_1^0 \\ a_{3,1} \leftarrow S_1^0 & a_{3,2} \leftarrow S_1^0 & a_{3,3} \leftarrow S_1^1 & a_{3,4} \leftarrow S_1^0 & a_{3,5} \leftarrow S_1^0 \\ a_{4,1} \leftarrow S_1^0 & a_{4,2} \leftarrow S_1^0 & a_{4,3} \leftarrow S_1^0 & a_{4,4} \leftarrow S_1^0 & a_{4,5} \leftarrow S_1^0 \\ a_{5,1} \leftarrow S_1^0 & a_{5,2} \leftarrow S_1^0 & a_{5,3} \leftarrow S_1^0 & a_{5,4} \leftarrow S_1^0 & a_{5,5} \leftarrow S_1^1 \end{bmatrix}$$

where each $a_{i,j}$ is chosen uniformly from all small elements of S_1^b.

At a first glance, this might seem like all we need to obfuscate a circuit. However it doesn't quite hold up against partial evaluation attacks which take advantage of the ordering of circuit representations or mixed input attacks where an adversary might choose matrices $A_{i,\mathrm{inp}(i)}$ inconsistently in the sense that the jth bit of the input might be treated as 1 the first time $\mathrm{inp}(i) = j$ and then 0 the next time $\mathrm{inp}(i) = j$. Defenses against both of these attacks rely heavily on randomization techniques involving the adding of "Bookends" and "Multiplicative Bundling" both of which, while important to constructing $i\mathcal{O}$, are highly technical and don't add much to understanding $i\mathcal{O}$. For our purposes we will acknowledge that our given scheme is not totally complete, but that fixes do exist and are described in detail in [Garg et al. (2013b)].

3.3 Obfuscation for Poly-sized Circuits

In this section we use the construction of indistinguishability obfuscation for Nick's Class along with Fully Homomorphic Encryption (FHE) to get obfuscation for polynomial sized circuits.

3.3.1 Fully Homomorphic Encryption

We won't get into much detail as to how FHE is constructed, instead we will describe what is expected of a FHE scheme. Like with our previously described forms of encryption, a FHE scheme is a 4-tuple of algorithms containing the standard key generation, encryption, and decryption functions. However, with FHE we include another function *evaluate* that takes as input a public key, a function, and a collection of ciphertexts and outputs the encrypted value of the function run on the encrypted input messages.

Definition 20. *Fully Homomorphic Encryption*

For message space $\mathcal{M}$, cipher space C, key space K, and circuit family $\mathcal{F}$ we define a Fully Homomorphic Encryption scheme to be the following functions:

- $KeyGen_{FHE} : \mathbb{Z} \to K \times K$
  Defined $KeyGen_{FHE}(\lambda) \to (pk, sk)$ *where* λ *is the security parameter and* pk, sk *are a public and secret key respectively*

- $Enc_{FHE} : K \times \mathcal{M} \to C$
  Defined $Enc_{FHE}(pk, m) \to c$ *meaning anyone with the public key can encrypt a message and get back a cipher text*

- $Dec_{FHE} : K \times C \to \mathcal{M}$
  Defined $Dec_{FHE}(sk, c) \to m$ *and it is expected that FHE schemes respect the public key encryption definition of correctness so* $Dec_{FHE}(sk, Enc_{FHE}(pk, m)) = m$ *and where the running time of* Dec_{FHE} *depends only on the security parameter* λ *and not the input.*

- $Eval_{FHE} : K \times \mathcal{F} \times C \times C \times \cdots \times C \to C$
  Defined $Eval_{FHE}(pk, f, c_1, c_2, \cdots, c_\ell) = c_f$ *for* $f \in \mathcal{F}$.

We say a FHE scheme is correct if for $c_1 = Enc_{FHE}(pk, m_1), \cdots, c_\ell = Enc_{FHE}(pk, m_\ell)$,

$$Dec(sk,\ Eval_{FHE}(pk, f, c_1, c_2, \cdots, c_\ell)) = f(m_1, m_2, \cdots, m_\ell)$$

As opposed to the notion of functional encryption introduced in the first chapter, with FHE decryption is all or nothing just like with classical encryption. A good use for FHE would be if Alice wants to have a large amount of data on a server, and be able to do computations on it. If Alice encrypts her data using a FHE scheme, she can then run Eval commands on the data, download the output of the evaluation command, and decrypt to find it's value. This works well because Alice doesn't need to download all the data onto her computer, decrypt, and then run the computation, she can decrypt her data before the overall computation is complete if she is curious about a sub result, and her data is kept secure since she is the only one able to decrypt with the secret key.

FHE would not be good for Alice if she wants to give her friend Bob access to a function of the data (like an average or the median) but does not want Bob to be able to see sub results of the computation. In fact, if encrypted using an FHE scheme, giving Bob the secret key would not only allow him to evaluate the average and then decrypt ($\mathrm{Dec}_{\mathrm{FHE}}(sk, \mathrm{Eval}(pk, \mathrm{average}, c_1, \cdots, c_\ell))$), but also decrypt any sub result of the computation he wants ($\mathrm{Dec}_{\mathrm{FHE}}(sk, c)$ for any c) including the raw data itself thus compromising the security of Alice's data.

3.3.2 Toward Poly-sized Circuit Obfuscation

For our construction of circuit obfuscation for polynomial sized circuits, we assume that we have an indistinguishability obfuscater for Nick's Class which we will denote $i\mathcal{O}_{\mathrm{NC}}$, and a fully homomorphic encryption scheme ($\mathrm{Setup}_{\mathrm{FHE}}$, $\mathrm{Encrypt}_{\mathrm{FHE}}$, $\mathrm{Eval}_{\mathrm{FHE}}$, Decrypt$_{\text{FHE}}$) where we can assume Decrypt$_{\text{FHE}}$ can be realized by a family of circuits in NC **FOR MORE INFORMATION SEE (CITATION TO JOSH'S THESIS)**. Last we also assume we have access to $\{U_\lambda\}$ where each $U_\lambda \in \{U_\lambda\}$ is a universal circuit polynomial in the size of λ and where $U_\lambda(C, m) = C(m)$ for circuit $C \in \{C_\lambda\}$ and message m.

Our scheme consists of two functions, Obfuscate and Evaluate. Informally, to obfuscate we will get FHE encryption of our circuit encoding for C which we will call g, then to evaluate the encrypted circuit on a message m we run the FHE evaluate function on a universal circuit U_λ with m built in to get $U_\lambda(g, m)$ which by correctness of FHE should decrypt to $C(m)$. More formally we construct Obfuscate and Evaluate as follows:

Obfuscate($\lambda, C \in \mathcal{C}_\lambda$) where $\mathcal{C}_\lambda$ are all circuits with size polynomial to λ.

1. Generate $(PK_{\text{FHE}}, SK_{\text{FHE}}) \leftarrow \text{Setup}_{\text{FHE}}(\lambda)$
2. Get FHE of circuit C, $g = \text{Encrypt}_{\text{FHE}}(PK_{\text{FHE}}, C)$
3. Define the program $P(\cdot) = \text{Decrypt}_{\text{FHE}}(SK_{\text{FHE}}, \cdot)$, the decryption algorithm with the secret key in it and then obfuscate it $\mathcal{P} = i\mathcal{O}_{\text{NC}}(P)$, note that we are relying on the fact that $\text{Decrypt}_{\text{FHE}} \in \text{NC}$
4. Public Parameters $PP = (\mathcal{P}, PK_{\text{FHE}}, g)$

Evaluate(PP, m) for valid message m.

1. Compute $e = \text{Eval}_{\text{FHE}}(PK_{\text{FHE}}, U_\lambda(\cdot, m), g) = U_\lambda(g, m) = \text{Encrypt}_{\text{FHE}}(PK_{\text{FHE}}, U_\lambda(C, m))$ by correctness of FHE
2. Output $\mathcal{P}(e) = i\mathcal{O}_{\text{NC}}(\text{Decrypt}_{\text{FHE}}(SK_{\text{FHE}}, e)) = C(m)$

At first glance the construction above might appear to yield indistinguishability obfuscation for poly-sized circuits, after all we have implicitly assumed our FHE scheme is correct and secure, so the circuit encrypted to get g should be "well hidden". Unfortunately the current scheme is not quite secure under our definition of indistinguishability obfuscation.

Suppose an adversary $\mathcal{A}$ gives us circuits $C_0, C_1 \in \{C_\lambda\}$ where

$$C_0(X = x_0||x_1) = x_0 \text{ or } x_1$$
$$\text{and}$$
$$C_1(X = x_0||x_1) = x_1 \text{ or } x_0$$

Where $C_0 = C_1$ since 'or' is commutative, so these are valid circuits to test if our scheme meets the indistinguishability obfuscation definition. Now let the encodings of C_0 and C_1 both be of length $\ell = p(\lambda)$ for some polynomial p, and let them be written in the same ordering specified above where the first k bits of the encoding of C_0 represents the term "x_0" and likewise the first $k' \approx k$ bits of the encoding of C_1 represent the term "x_1". Then if we run Obfuscate(C_0) $\rightarrow PP_0$ and Obfuscate(C_1) $\rightarrow PP_1$ and give PP_b to the adversary $\mathcal{A}$ for $b \leftarrow \{0,1\}$ chosen uniformly, consider the following attack.

Upon receiving $PP_b = (\mathcal{P}_b, PK^b_{\text{FHE}}, g_b)$, $\mathcal{A}$ wants to differentiate PP_0 from PP_1 by which bit appears first in the encoding of their associated circuits. To do this $\mathcal{A}$ generates a mask w of length ℓ such that the first k bits of the mask are all 1 and the last $\ell - k$ bits are all 0 so that it's of the form $w = 11\cdots100\cdots0$. Then $\mathcal{A}$ encrypts the mask $w_{\text{Enc}} = \text{Encrypt}_{\text{FHE}}(PK^b_{\text{FHE}}, w)$ and then multiplies the encrypted circuit g_b by the encrypted mask by getting g'_b =$\text{Eval}_{\text{FHE}}(PK_{\text{FHE}}, \prod, w_{\text{Enc}}, g_b)$ and then for message $m = 01$ output $\mathcal{P}_b(\text{Eval}_{\text{FHE}}(PK_{\text{FHE}}, U_\lambda(\cdot, m), g'_b) \to b'$.

Note that if $b' = 0$ then x_0 appears first in the encoding giving away that $\mathcal{A}$ received an obfuscation of C_0, and similarly $b' = 1$ implies that $\mathcal{A}$ received an obfuscation of C_1. Thus this construction doesn't meet our security definition of indistinguishability obfuscation. To fix this construction we need a way of ensuring that evaluation is run on g and *only* g without any alterations. In order to do this we need another cryptographic primitive.

3.3.3 Low Depth Non Interactive Zero Knowledge Proofs

Non interactive zero knowledge (hence forth abbreviated NIZK) proofs are a cryptographic primitive between a *prover* and a *verifier*. We will not give any sort of formal construction of how NIZK proofs are made, rather we focus on what they ensure. Informally, we expect a NIZK proof to work as follows:

Let Peggy be our prover and Victer be our verifier. Given a statement and a witness to that statement a Peggy should be able to generate a proof π such that when π is given to Victer, he is convinced that π is a valid proof and yet anyone given π should be unable to use π to gain any information about the witness to the statement.

More formally a NIZK proof is a one time interaction between a prover and verifier of sending a proof π from the prover to the verifier where π has the following properties:

- Perfect Completeness: A proof system is perfectly complete if an honest prover can generate a proof π that x exists in language L such that an honest verifier is always convinced that $\pi \implies x \in L$.

- Statistical Soundness: A proof system is sound if it is infeasible for a prover to generate a false proof that convinces an honest verifier

- Computational Zero Knowledge: A proof system is computational zero-knowledge if the proof does not reveal any information about the witness to any adversary.

- Low Depth: We say a proof is low depth if the verifier can be implemented in NC.

Rough details into how a low depth NIZK proof system could be constructed can be found in appendix B of [Garg et al. (2013b)], for our purposes we only need to know that such proof systems exist and guarantee the above properties.

3.3.4 Obfuscation for Poly-Sized Circuits

How might we use low depth NIZK proofs to fix our previous construction of $i\mathcal{O}_p$? As stated at the end of section 3.3.2, we need a way of ensuring that our decryption program only decrypts cipher texts where the message has been *fully* evaluated by our encrypted circuit g.

We begin our edits of the previous scheme by first revising how our program P runs. We define P to be a function that not only takes a cipher text e as before, but now also takes a proof π and a message m such that for $(PK_{\text{FHE}}, SK_{\text{FHE}}) \leftarrow \text{Setup}_{\text{FHE}}(\lambda)$:

$P(e, m, \pi) =$

- First we check if π is a valid low-depth proof that

$$e = \text{Eval}_{\text{FHE}}(PK_{\text{FHE}}, U_\lambda(\cdot, m), g)$$

- If the check fails output $\perp$ indicating the inputs are invalid; otherwise output $\text{Decrypt}_{\text{FHE}}(e, SK_{\text{FHE}})$

In essence, the evaluator acts as a prover for the statement "Did you appropriately generate your cipher text?" and the program P acts as the verifier for proofs π of that statement. Here "appropriately generated" means the entire encrypted circuit g was run on the specified message m. Since we know indistinguishability obfuscation is the best possible obfuscation we can assume that any adversary with $i\mathcal{O}_{\text{NC}}(P)$ can't alter the hard coded g in P.

The only other thing we need to check is that our program $P \in$NC so that we can use $i\mathcal{O}_{\text{NC}}$ on it. As before we know/assume Decrypt$\in$NC, and if we use a low-depth NIZK proof system for step 1 then the verifier$\in$NC so $P \in$NC. Thus we have constructed $i\mathcal{O}_P$ an indistinguishability obfuscater for poly-size circuits.

Chapter 4

Functional Encryption

4.1 Definition

Now that we have built up an understanding of multilinear maps and circuit obfuscation, we are ready to talk about functional encryption. Recall the following definition of functional encryption where secret keys are generated for different functionalities.

Definition 21. *Functional Encryption*

A functional encryption scheme for a class of functions $\mathcal{F} = \mathcal{F}(\lambda)$ over message space $\mathcal{M} = \mathcal{M}_\lambda$ consists of four algorithms $\mathcal{FE}$ = { Setup, KeyGen, Encrypt, Decrypt} where:

- *$Setup_{FE}(1^\lambda) \to (PP, MSK)$ takes security parameter and outputs Public Parameter and Master Secret Key*
- *$KeyGen_{FE}(MSK, f) \to SK_f$ takes Master Secret Key and a function $f \in \mathcal{F}$ and outputs a corresponding secret key SK_f*
- *$Encrypt_{FE}(PP, m) \to c$ takes Public Parameter and a message $m \in \mathcal{M}$ and outputs cipher text c*
- *$Decrypt_{FE}(SK_f, c) \to m'$ takes secret key and cipher text and outputs message m'*

We say a functional encryption scheme is correct if

$$\forall m \in \mathcal{M},\ Decrypt(SK_f, Encrypt(PP, m)) = f(m)$$

4.2 Construction

Here we give a simplified construction of the functional encryption scheme described in the previous section, a more detailed construction using non interactive zero knowledge proofs can be found in Garg et al. (2013b). Here we assume that we have access to a public key encryption scheme with algorithms ($\text{Setup}_{\text{PKE}}$, $\text{Encrypt}_{\text{PKE}}$, $\text{Decrypt}_{\text{PKE}}$), then

- $\text{Setup}_{FE}(\lambda)$: Takes security parameter λ and generates
$$(PP, MSK) \leftarrow \text{Setup}_{\text{PKE}}(1^\lambda)$$
- KeyGen $_{\text{FE}}$(MSK,f): Outputs an obfuscation $SK_f := i\mathcal{O}(P_f(MSK, \cdot))$ for the program
$P_{f,MSK}(c, \pi)$=
  - Check π is a valid proof that $\exists m, r \in \mathcal{M}$ such that c =$\text{Encrypt}_{\text{PKE}}(PK, m; r)$ where r is used to account for the randomization of our encryption function
  - if the check fails output $\perp$ otherwise output $f(\text{Decrypt}_{\text{PKE}}(MSK, c))$
- Encrypt $_{\text{FE}}$(PP, m)$\in \{0,1\}^n$): Outputs $c =$ Encrypt $_{\text{PKE}}(PP, m)$ and π a NIZK proof that c was produced as just specified.
- Decrypt $_{\text{FE}}(SK_f, c)$: Outputs m' by running $SK_f(c, \pi)$.

It might take some rereading of the simplified construction above to understand what is going on. Essentially, our SK_f is an obfuscated program which first fully decrypts a cipher text, then outputs the result of running f on the decrypted message. The idea of fully decrypting the cipher text and then evaluating the function on the decrypted message is why we need obfuscation for this scheme to work since working with an obfuscated program should give no more information than just its input and output behaviors (at-least as we said before this is how indistinguishability obfuscation is often treated).

4.3 Application

In this section we talk a little more about use cases for functional encryption. Here we will discuss high level implementations of functional encryption for uses in the real world. As of now software, such as 5gen-C which we will be discussing in chapter 5, are limited to encrypting functions with low multi-linearity, we will get back to this later, but the point is some of these examples can not be feasible instantiated with the current notion of graded encoding schemes.

4.3.1 Sealed-Bid First-Price Auction

Here we describe how functional encryption can be used to implement fair and secure sealed-bid first-price auctions. A sealed-bid first-price (hereby referred to as SBFP) auction is an auction where each bidder submits a single bid where the amount is kept secret from the other bidders. Once all bids are submitted, the highest bidder pays their price and wins the auction. Often the seller at these auctions will have a predetermined floor price where if the winning bid is below this floor, the item doesn't sell.

A common setting for SBFP auctions is when firms apply for government contracts, in this case a government agency announces that they are looking to contract for a certain service such as road repair, at which points firms bid the amount they would expect to be paid to do the road repairs, and then once all bids are submitted the government is required by law to contract to the "lowest responsible" bidder which means the lowest *price*, qualified firm.

The auction setting we will set up a functional encryption scheme for is one where *only* the highest bid should be known to the seller. Perhaps we could consider the case of a sensitive artist selling their own artwork at an auction, or friends bidding on another friend's artwork, some of which don't particularly like the art, but do not want to appear rude by bidding extremely low or not at all. Each of these situations is perfect for functional encryption since we only want the seller to be able to learn what the encrypted value of the highest valued cipher-text is.

We assume that we have access to a valid and binding digital signature scheme consisting of the polynomial time algorithms

$$\big(\mathrm{Gen}_{\mathrm{sign}}(\lambda) \to (PK_{\mathrm{sign}}, SK_{\mathrm{sign}}),\ \mathrm{Sign}(SK_{\mathrm{sign}}, m) \to \sigma,\ \mathrm{Verify}_{\mathrm{sign}}(PK_{\mathrm{sign}}, m, \sigma) \to \{0,1\}\big)$$

Auction Initialization: The auction coordinator, an impartial third party (probably some sort of web app), is chosen and initializes the auction as follows:

- Run $\mathrm{Setup}_{\mathrm{FE}}(\lambda) \to (PP, MSK)$ and make PP public

Bidder Initialization: Each bidder is expected to send their bid as an encryption of the bid, and encryption of a signature on that bid, the public key for the signature, and the bidder's name. More formally we expect a bidder to:

- Chosen a bid amount which we denote b
- Run $\mathrm{Gen}_{\mathrm{sign}}(\lambda) \to (PK_{\mathrm{sign}}, SK_{\mathrm{sign}})$
- Generate a signature: $\mathrm{Sign}(SK_{\mathrm{sign}}, b) \to \sigma$
- Run $\mathrm{Encrypt}_{\mathrm{FE}}(PP, b) \to \mathbf{c}_{\mathrm{bid}} := (c_{\mathrm{bid}}, \pi_{\mathrm{bid}})$
- Run $\mathrm{Encrypt}_{\mathrm{FE}}(PP, \sigma) \to \mathbf{c}_{\mathrm{sign}} := (c_{\mathrm{sign}}, \pi_{\mathrm{sign}})$
- To bid, send $m = (\mathbf{c}_{\mathrm{bid}}, \mathbf{c}_{\mathrm{sign}}, PK_{\mathrm{sign}}, \text{Bidder's Name})$ to the seller.

We require the bidder's signature to ensure that the bidder is honest about their bid amount (we will give more detail about this when describing the Auction phase), but also to ensure that no one can "frame" a bidder by potentially bidding an exorbitant amount in someone else's name.

Program Initialization: The programs for obfuscation or to be used in functional encryption are as follows:

- $G(x, y)$ is a circuit that takes two integers encoded in binary and outputs 1 if

$$x > y$$

- $E(x, y)$ is a circuit that takes two integers encoded in binary and outputs 1 if

$$x == y$$

Auction: The for G, E and $\text{Verify}_{\text{sign}}$ are distributed to bidders and seller, if a participant agrees with the two programs they sign each encoding and send the encoding and signature to the auction coordinator (hereby AC) who checks that all the signatures are valid. Then

- AC Generates $\text{KeyGen}_{\text{FE}}(MSK, G) \rightarrow SK_G$
- AC Generates $\text{KeyGen}_{\text{FE}}(MSK, E) \rightarrow SK_E$
- AC Generates $\text{KeyGen}_{\text{FE}}(MSK, \text{Verify}_{\text{sign}}) \rightarrow SK_{\text{Verify}}$
- AC sends SK_C and SK_{Verify} to the seller
- The seller runs the following FindWinner algorithm on $M = [m^1, m^2, \cdots, m^n]$ where m^{n+1} is the seller's floor encoded as specified in Bidder Initialization, Note that for the sake of notation superscripts indicate the index in M and subscripts describe the piece of m^i.

  $\text{FindWinner}([m^1, m^2, \cdots, m^n, m^{n+1}])$

```
- $i = 2$; max = 1; tie = False
- While $i \leq n + 1$:
    * if $\text{Decrypt}_{\text{FE}}(SK_G, \mathbf{c}^i_{\text{bid}}, \mathbf{c}^{\max}_{\text{bid}}) == 1$ # if $i$th bid is greater than current maximum bid
        · max = $i$ # the bid at $i$ is the new maximum bid
        · tie = False
    * if $\text{Decrypt}_{\text{FE}}(SK_E, \mathbf{c}^i_{\text{bid}}, \mathbf{c}^{\max}_{\text{bid}}) == 1$ # if $i$th bid is equal to current maximum bid
        · tie = True
    * i++
- if $\text{Decrypt}_{\text{FE}}(SK_{\text{Verify}}, PK^{\max}_{\text{sign}}, \mathbf{c}^{\max}_{\text{bid}}, \mathbf{c}^{\max}_{\text{sign}}) == 1$
    * if tie == True
        · print "There was a tie, bid again"
    * else return max
- else return $\perp$
```

  The above program uses SK_G to find the maximum valued bid, and SK_E to see if there are multiple maximal bids in which case there will need to be some type of tie breaker, probably in the form of another round of bids.

- Since $i\mathcal{O}$ is limited to binary outputs, we can not get the value of the bid out of the cipher text directly. Instead we have required bidders to also encrypt a signature so that we can use SK_{Verify} to verify the validity of the signature and commit the bidder to their encrypted bid. For the seller to learn the value of the maximum bid, the seller must tell the winning bidder (whose name is encoded in m^{max}) that they are the winner and ask them to send over the value of their bid b. The seller then needs to verify that the bid amount b is in fact the same amount that was encrypted in $\mathbf{c}_{\text{bid}}^{\text{max}}$ so the seller runs

  $$\text{Decrypt}_{\text{FE}}\big(SK_{\text{Verify}}, PK_{\text{sign}}^{\text{max}}, \text{Encrypt}_{\text{FE}}(PP, b), \mathbf{c}_{\text{sign}}^{\text{max}}\big) \to v$$

  Note: that we encrypt b so that it is in the proper format for using SK_{Verify}

  If $v == 1$ this means that b is the amount of the maximum bid and so the seller knows with confidence that the winning bidder will pay the amount the bid. If $v == 0$ then the bidder has submitted a bid inconsistent with the bid encoded in $\mathbf{c}_{\text{bid}}^{\text{max}}$ at which point the max bidder will probably need to either submit a consistent bid value or be penalized in some way.

Chapter 5

A Brief Introduction to the 5-GenC library

delete this eventually Carmer et al. (2017)

5.1 The DSL

5.2 Circuits and Branching Programs

5.3 Base and MMaps

Chapter 6

Experiments

6.1 Comparison Circuit

6.2 Runtime Evaluation

Chapter 7

Conclusion

References

Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S., & Yang, K. (2001). On the (im)possibility of obfuscating programs. In J. Kilian (Ed.), *Advances in Cryptology — CRYPTO 2001*, (pp. 1–18). Berlin, Heidelberg: Springer Berlin Heidelberg.

Boneh, D., Sahai, A., & Waters, B. (2011). Functional encryption: Definitions and challenges. In Y. Ishai (Ed.), *Theory of Cryptography*, (pp. 253–273). Berlin, Heidelberg: Springer Berlin Heidelberg.

Carmer, B., Malozemoff, A. J., & Raykova, M. (2017). 5gen-c: Multi-input functional encryption and program obfuscation for arithmetic circuits. Cryptology ePrint Archive, Report 2017/826. `https://eprint.iacr.org/2017/826`.

Garg, S., Gentry, C., & Halevi, S. (2013a). Candidate multilinear maps from ideal lattices. In T. Johansson, & P. Q. Nguyen (Eds.), *Advances in Cryptology – EUROCRYPT 2013*, (pp. 1–17). Berlin, Heidelberg: Springer Berlin Heidelberg.

Garg, S., Gentry, C., Halevi, S., Raykova, M., Sahai, A., & Waters, B. (2013b). Candidate indistinguishability obfuscation and functional encryption for all circuits. In *Proceedings of the 2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, FOCS '13, (pp. 40–49). Washington, DC, USA: IEEE Computer Society.

Goldwasser, S., & Rothblum, G. N. (2007). On best-possible obfuscation. In *Proceedings of the 4th Conference on Theory of Cryptography*, TCC'07, (pp. 194–213). Berlin, Heidelberg: Springer-Verlag. `http://dl.acm.org/citation.cfm?id=1760749.1760765`