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 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 immediately 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, so that every possible key can't be attempted by an adversary; and chosen randomly, so they are nearly impossible to guess.

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 $Dec_s k(c) \to m$ 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 Bob would have to run $\text{Gen}(n) \to pk, sk$ and then send pk to Alice. Then Alice gets $c :=\text{Enc}_{pk}(m)$ and sends c over to Bob. Finally Bob gets $m' :=\text{Dec}_{sk}(c)$. 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
$\text{Gen}(\lambda) \to (sk, pk)$		
	$\xrightarrow{pk}$	
		$\text{Enc}_{pk}(m) \to c$
	$\xleftarrow{c}$	
$m' := \text{Dec}_{pk}(c)$		

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 and Identity Based Encryption. To see how these primitives are sub cases of Functional Encryption. Lets formalize our idea of a Functional Encryption Scheme.

To define a Functional Encryption Scheme, we must first define a way of describing what a cipher text can be decrypted to.

Think of something better than case space, it's confusing with the notation for a cipher space. The paper calls it a key space K but that's also confusing. Change later, keep in mind now.

Definition 4 (Functionality)**.** *Given a case space $C_{ase} \cup \{\epsilon\}$, message space M we define the functionality F to be*

$$F : C_{ase} \times M \to M$$

Functionality describes what the possible outputs are. In public key encryption, knowing the secret key sk allows for the message to be read in full, but without the secret key, only the length of the message can be discerned from the cipher text. To write this in the syntax of a functionality we define

$$F(c, m) = \begin{cases} x \text{ if } c = 1 \\ \text{length}(x) \text{ if } c = \epsilon \end{cases}$$

The only functionality of public key encryption is fully decoding the message so this is our primary case ($c = 1$), however we also account for the information learned without the public key which is an unavoidable rather than built in case ($c = \epsilon$).

Definition 5 (Functional Encryption Scheme)**.** *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) $\to (pk, mk)$, generates a public key and master key

$$Gen : K \times C \to K$$

Defined for $c \in C_{ase}, mk \in K$; to be $Gen(mk, c) \to sk$ which is kept secret and is the secret key of functionality c.

$$Enc : K \times M \to C_{ipher}$$

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

$$Dec : K \times C_{ipher} \to M$$

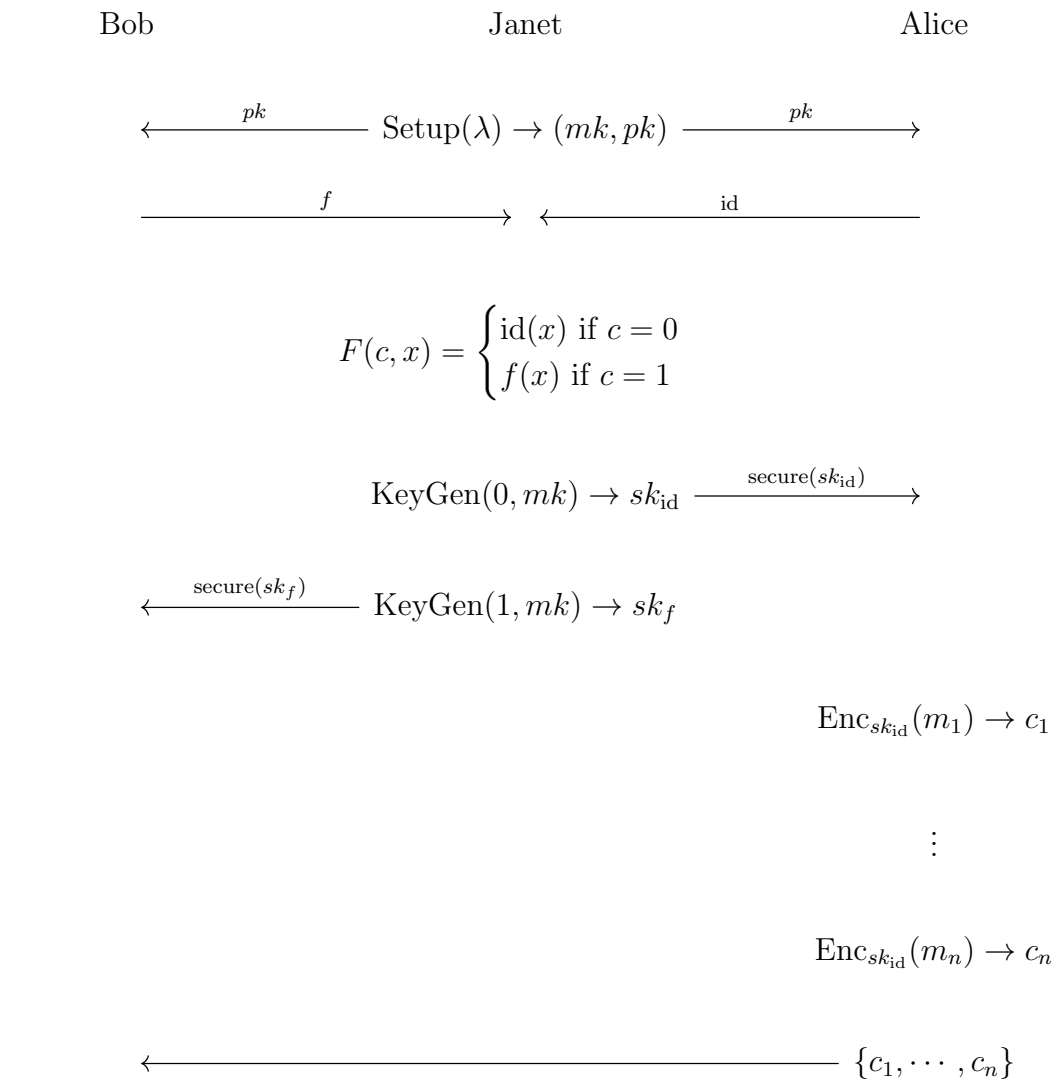
Defined for $ek \in K$ and $c \in C_{ipher}$; to be $Dec(k, c) \to n$ where $n = F(k, m)$ for some functionality F.

If the notion of a Functionality was confusing before, the use of it in generating the secret key should make it clear.

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 $\text{Setup}(\lambda) \rightarrow (mk, pk)$ and publishes the public key, and also runs $\text{Keygen}(mk, 0) \rightarrow sk_{id}$ and $\text{Keygen}(mk, 1) \rightarrow sk_f$ for functionality:

$$F(c, m) = \begin{cases} f(m) \text{ if } k = 0 \\ \text{id}(m) \text{ if } k = 1 \end{cases}$$

Then Janet securely sends sk_{id} to Alice and sk_f to Bob. Now Alice encrypts her text files $\text{Enc}_{pk}(m_1) \rightarrow c_1$,$\text{Enc}_{pk}(m_2) \rightarrow c_2, \cdots$,$\text{Enc}_{pk}(m_n) \rightarrow c_n$, and sends $\{c_1, c_2, \cdots, c_n\}$ to Bob for storage. Bob then runs $\text{Dec}_{sk_f}(c_1) \rightarrow b_1, \cdots, \text{Dec}_{sk_f}(c_n) \rightarrow b_n$ and allows storage of all c_i that indicate the word "cat" is not being stored. This interaction is illustrated in the diagram below:



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 **CITATION NEEDED**, Identity Based Encryption **CITATION NEEDED**, and Multi-Party Input Functional Encryption **CITATION NEEDED** which we will actually work with later on.

1.2 Obfuscation

1.2.1 Black Box Obfuscation

In short, 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 was first defined by **(INSERT CITATION)**. 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 of the that circuit (we call this table an Oracle since).

Definition 6. *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 $\mathcal{O}_c(\cdot)$.*

FIND A BETTER DEFINITION OF BLACK BOX OBFUSCATION

While Black Box Obfuscation has been proven impossible for general circuits and programs (**insert citation and perhaps some more details as to what exactly has been proven impossible**)

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 $\mathcal{O}(\text{Enc}_{sk}(\cdot) \to \text{Enc}(\cdot)$ where $\text{Enc}(\cdot)$ 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 **CITATION**, 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.

Definition 7 (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 the two circuits are 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 as good as the best possible obfuscation, the proof is short and eloquent so we will show it here but it was originally formulated by (**INSERT CITATION HERE**)

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. This could lead to issues in the future and there is lots of room for work to be done in quantifying degrees of Obfuscation.

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 8. *Bilinear Maps Let G_1, G_2, G_3 be cyclic groups of prime order p. Then we say a map $e : G_1 \times G_2 \to G_3$ 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 $g_1 \in G_1$ and $g_2 \in G_2$, $e(g_1, g_2) = g_3$ a generator of G_3.*

1.3.2 Intuition

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

Suppose Alice, Bob and Carol wish to agree upon a single secret key for the three of them. To do so Alice would run $\text{InstGen}(\lambda, 2) \to (\mathbf{params}, g_1)$ where g_1 generates G_1. Then she would choose a secret $a \in \mathbb{Z}_p$ and broadcast $(e \in \mathbf{params}, g_1, pk_a = g_1^a)$. 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 2-Linear 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*, since we usually can't prove there are no ways to break these assumptions.

To formalize a hardness assumption about the security of bilinear maps we first define the following problems.

Definition 9. *Bilinear Computational Diffie-Hellman Problem*

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 $a, b, c, \in \mathbb{Z}_p$ 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 10. *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 $g_1^\beta \; \forall \beta \in \mathbb{Z}_p$. 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 **CITATION?**, but these arguments are outside the scope of this thesis. It is enough for us that up to now cryptographers and mathematicians have been unable to give efficient solutions to solving these problems.

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 on encoded secrets securely. Specifically, 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 securely 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 11. *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.*

As before we can think of $\alpha \cdot g_i$ to be g_i^α for $g_i \in G_i$ and $\alpha \in \mathbb{Z}_p$. 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. The first is just an extension of the bilinear discrete logarithm problem.

Definition 12. *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.*

and the other is

Definition 13. *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

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

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, and then we'll build up to the actual definition from there.

Definition 14. *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

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

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. 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 tiered structure of the scheme. In our simplified definition, encodings are unique and of the form g_i^α. In practice there are sets of 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 15.
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 **CITATION**, 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. Also

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

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

2.2.3 Definition

If this is your first time reading this paper, it is recommended to read the following definition, but not expect to understand much of it, then after reading the intuition and construction section, reread the formal definition and it should be much clearer.

Definition 16. *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.
- **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
- **Zero-test**: takes as inputs $\mathcal{S}$ and u, outputs 1 if $u \in S_k^{(0)}$ and 0 otherwise.
- **Extraction**:?????

2.2.5 Hardness Assumptions

Draw parallels between the Bilinear CDD assumption and the one made in GGH.

Chapter 3

Indistinguishability Obfuscation

3.1 Notes

3.2 Construction

3.3 Usage, Limitations, and Goals

Chapter 4

Multi-Party Input Functional Encryption

4.1 Scheme

4.2 Construction

4.3 Limitations and Goals

Chapter 5

A Brief Introduction to the 5-GenC library

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

Angel, E. (2000). *Interactive Computer Graphics : A Top-Down Approach with OpenGL*. Boston, MA: Addison Wesley Longman.

Angel, E. (2001a). *Batch-file Computer Graphics : A Bottom-Up Approach with QuickTime*. Boston, MA: Wesley Addison Longman.

Angel, E. (2001b). *test second book by angel*. Boston, MA: Wesley Addison Longman.

Deussen, O., & Strothotte, T. (2000). Computer-generated pen-and-ink illustration of trees. *"Proceedings of" SIGGRAPH 2000*, (pp. 13–18).

Fisher, R., Perkins, S., Walker, A., & Wolfart, E. (1997). *Hypermedia Image Processing Reference*. New York, NY: John Wiley & Sons.

Gooch, B., & Gooch, A. (2001a). *Non-Photorealistic Rendering*. Natick, Massachusetts: A K Peters.

Gooch, B., & Gooch, A. (2001b). *Test second book by gooches*. Natick, Massachusetts: A K Peters.

Hertzmann, A., & Zorin, D. (2000). Illustrating smooth surfaces. *Proceedings of SIGGRAPH 2000*, *5*(17), 517–526.

Jain, A. K. (1989). *Fundamentals of Digital Image Processing*. Englewood Cliffs, New Jersey: Prentice-Hall.

Molina, S. T., & Borkovec, T. D. (1994). The Penn State worry questionnaire: Psychometric properties and associated characteristics. In G. C. L. Davey, & F. Tallis (Eds.), *Worrying: Perspectives on theory, assessment and treatment*, (pp. 265–283). New York: Wiley.

Noble, S. G. (2002). *Turning images into simple line-art*. Undergraduate thesis, Reed College.

Reed College (2007). Latex your document. `http://web.reed.edu/cis/help/LaTeX/index.html`

Russ, J. C. (1995). *The Image Processing Handbook, Second Edition*. Boca Raton, Florida: CRC Press.

Salisbury, M. P., Wong, M. T., Hughes, J. F., & Salesin, D. H. (1997). Orientable textures for image-based pen-and-ink illustration. *"Proceedings of" SIGGRAPH 97*, (pp. 401–406).

Savitch, W. (2001). *JAVA: An Introduction to Computer Science & Programming*. Upper Saddle River, New Jersey: Prentice Hall.

Wong, E. (1999). *Artistic Rendering of Portrait Photographs*. Master's thesis, Cornell University.