

- 213). Berlin, Heidelberg: Springer-Verlag. <http://dl.acm.org/citation.cfm?id=1760749.1760765>
- Katz, J., & Lindell, Y. (2007). *Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series)*. Chapman & Hall/CRC.
- Kilian, J. (1988). Founding cryptography on oblivious transfer. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, (pp. 20–31). New York, NY, USA: ACM. <http://doi.acm.org/10.1145/62212.62215>
- Langlois, A., Stehle, D., & Steinfeld, R. (2014). Gghlite: More efficient multilinear maps from ideal lattices. Cryptology ePrint Archive, Report 2014/487. <https://eprint.iacr.org/2014/487>.

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

## 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.
- Barrington, D. A. (1986). Bounded-width polynomial-size branching programs recognize exactly those languages in  $nc1$ . In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, STOC '86, (pp. 1–5). New York, NY, USA: ACM. <http://doi.acm.org/10.1145/12130.12131>
- 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>.
- Coron, J.-S., Lepoint, T., & Tibouchi, M. (2013). Practical multilinear maps over the integers. In R. Canetti, & J. A. Garay (Eds.), *Advances in Cryptology – CRYPTO 2013*, (pp. 476–493). Berlin, Heidelberg: Springer Berlin Heidelberg.
- Gancher, J. (2016). Fully homomorphic encryption. Reed College Electronic Theses. [https://rdc.reed.edu/c/etheses/s/r?\\_pp=20&query=fully%20homomorphic%20encryption&s=5dce75f4a8c2af56899bce985b0e871fa2c150d5&p=1&pp=1](https://rdc.reed.edu/c/etheses/s/r?_pp=20&query=fully%20homomorphic%20encryption&s=5dce75f4a8c2af56899bce985b0e871fa2c150d5&p=1&pp=1)
- 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–

Approved for the Division  
(Mathematics)

---

Dylan McNamee

as well as a method that uses functional encryption referred to in 5gen-C as CMR which is used as default. We will not go much into these obfuscation methods but explanations can be found in [Carmer et al. (2017)].

For multi-input functional encryption (hence forth MIFE), `mio mife` currently supports single key secret key MIFE which combines the Setup and KeyGen algorithms described in the generalized functional encryption described in chapter 4. We define sk-MIFE to be the following three polynomial time algorithms:

- $\text{Setup}(\lambda, C) \rightarrow (PP, SK_C)$ : Takes as input a security parameter and circuit description and outputs a secret key and evaluation key.
- $\text{Enc}(PP, i, m) \rightarrow c$ : Takes as input a secret key, input slot, and message; and outputs a cipher text. For an MIFE of “ $x_0 > x_1$ ” the input slot 0 would indicate that the message should be encrypted to take the place of “ $x_0$ ” while an input slot of 1 would indicate to encrypt for “ $x_1$ ”.
- $\text{Dec}(SK_C, [c_0, c_1, \dots, c_n]) \rightarrow b$ : Takes as input an evaluation key and an ordered list of cipher texts and outputs a bit  $b = C(m_0, m_1, \dots, m_n)$ .

The major differences between sk-MIFE and generalized MIFE is that the cipher texts are generated specific to an evaluation circuit. This means that ciphertexts can *only* be functionally decrypted for the specific  $C$  input into the setup algorithm. For example, if  $G$  is a circuit that evaluates  $x_0 > x_1$ , all cipher texts generated from  $\text{Enc}(PP, i, m) \rightarrow c_i$  for  $SK_G \leftarrow \text{Setup}(\lambda, G)$  will only ever be able to be decrypted to evaluating  $G$ , this means all functionalities that we might want to decrypt to must be built into the circuit initially entered into the Setup algorithm.

## 5.2 Experiments

a circuit functionality then run compile that program with `cxs` and a specific input size to get a circuit of the functionality described in the DSL that expects inputs of the specified size.

For example suppose we want a circuit for greater than that compares two inputs each length 1, as well as a circuit for greater than that compares two inputs each of length 2. Without `cxs` we would need to code up two different circuits in the form of a `acirc` program; but with `csx` we can describe greater than circuits of arbitrary input size and then compile it with `cxs` for input sizes 1 and 2 in order to get the two `acirc` files we want. To compile a circuit simply write a the circuit in Haskell (using special 5Gen-C packages), and then run `./cxs compile .acirc filename`.

For example, a circuit that compares two bits (i.e.  $x_0 > x_1$  where  $x_0, x_1$  are each either 1 or 0) compiles to the following `.acirc` file:

```
0 input 0 : 3
1 input 1 : 3
2 const 0 : 2
3 sub 2 1 : 1
4 mul 0 3 : 1
5 add 0 1 : 1
6 mul 0 1 : 2
7 add 6 6 : 1
8 sub 5 7 : 1
```

`.acirc` files are for the `.mio` program, but the above `.acirc` file represents the circuit  $x_0 > x_1$  similar to a branching program, but with addition, multiplication, and subtraction (otherwise known as bitwise ‘OR’, ‘AND’, and ‘XOR’) built in.

### 5.1.3 mio

5Gen-C contains a command that allows for implementing **multi-input** functional encryption and **obfuscation**, called `mio`. Obfuscation with `mio` is the same as we have described, in chapter 3. `mio obf` takes as input a circuit (in the form of a `.acirc` program) and outputs an obfuscated circuit.

Right now, `mio obf` supports a few different obfuscation techniques that allow for obfuscation of circuits directly without first needing to convert them to matrix branching programs. Such techniques use methods similar to the ones we described in chapter 3, however instead of using matrices, entries into boolean expressions are encoded under a graded encoding scheme, and different randomization techniques are used to ensure security from mixed input or partial evaluation attacks. 5Gen-C allows for obfuscation with previously established methods such as Zimmerman (referred to as Zim) obfuscation and Lin obfuscation, but also derived a few new methods such as the so called Linnerman method (a combination of techniques from Zim and Lin)

## Acknowledgements

# Chapter 5

## 5Gen-C

While we have shown that multilinear maps, obfuscation, and functional encryption are all *possible*; we have yet to discuss their feasibility. One such tool currently in development for actually using the primitives we have discussed is the 5Gen-C framework [Carmer et al. (2017)]. We begin this chapter with a brief overview of what tools are available in 5Gen-C and then go on to discuss experiments we have run to test efficiency of different encrypted circuits.

### 5.1 5Gen-C Overview

5Gen-C offers three major tools related to the primitives we have been discussing. `libmmap` which allows for easy implementation of multilinear maps, `mio` for obfuscating circuits as well as implementing functional encryption, and `CXS` which compiles circuits described in Haskell into a domain specific language for branching programs (`.acirc`). We describe the usage of these tools with a little more detail below.

#### 5.1.1 libmmap

`libmmap` is a library that allows for implementation of the GGHLite [Langlois et al. (2014)] with `libgghlite` and defaults to CLT [Coron et al. (2013)] with `libclt`. We will not dwell on `libmmap` since for our purposes we only use it as a backend library for the `mio` program for functional encryption described later. Essentially `libmmap` gives an interface for using multilinear maps with commands for getting the public parameter or secret key from instantiation, as well as for encoding plaintext elements, running operations on them, and zero testing the results. For more detail see [Carmer et al. (2017)].

#### 5.1.2 CXS

5Gen-C contains a program circuit-synthesis (hereby `cxs`), for compiling and optimizing circuits from a domain specific language (DSL) embedded in Haskell to `acirc` files ready for obfuscation. We use `cxs` to write Haskell like programs that describe

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  $c_{\text{bid}}^{\text{max}}$  so the seller runs

$$\text{Decrypt}_{\text{FE}}(SK_{\text{Verify}}, \text{Encrypt}_{\text{FE}}(PK_{\text{FE}}, PK_{\text{sign}}^{\text{max}}), \text{Encrypt}_{\text{FE}}(PP, b), c_{\text{bid}}^{\text{max}}) \rightarrow v$$

Note: that we encrypt  $b$  and  $PK_{\text{sign}}^{\text{max}}$  so that it is in the proper format for using  $SK_{\text{Verify}}$

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 they initially bid. If  $v == 0$  then the bidder has submitted a bid inconsistent with the bid encoded in  $c_{\text{bid}}^{\text{max}}$  at which point the max bidder will probably need to either submit a consistent bid value or be penalized.

A few things about this scheme should be noted. While bid amounts can only be revealed by having the bidder send in their bid after the auction, an ordering of *all* the bids can easily be found by using  $SK_G$  to sort the cipher texts by bid amount. This means that a seller can find out who bid not only the highest, but the lowest as well. This means the seller can learn more than just the identity of the highest bidder from the auction, which may not be an issue for users, but should be understood by anyone looking to utilize this auction protocol.

## Abstract

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

- 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  $SK_G, SK_E$  and  $SK_{\text{Verify}}$  to the seller
- The seller runs the following FindWinner algorithm on  $[m^1, m^2, \dots, m^n, m^{n+1}]$  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 the list and subscripts describe the piece of  $m^i$  we are focusing on (i.e.  $c_{\text{bid}}$  or  $c_{\text{sign}}$ ).  
FindWinner( $[m^1, m^2, \dots, m^n, m^{n+1}]$ )
  - $i = 2$ ;  $\text{max} = 1$ ;  $\text{tie} = \text{False}$
  - While  $i \leq n + 1$ :
    - \* if  $\text{Decrypt}_{\text{FE}}(SK_G, c_{\text{bid}}^i, c_{\text{bid}}^{\text{max}}) == 1$  # if  $i$ th bid is greater than current maximum bid
      - $\text{max} = i$  # the bid at  $i$  is the highest bid seen so far
      - $\text{tie} = \text{False}$  # this bid is **maximal** so if it “wins” there will need to be a tie breaker
    - \* if  $\text{Decrypt}_{\text{FE}}(SK_E, c_{\text{bid}}^i, c_{\text{bid}}^{\text{max}}) == 1$  # if  $i$ th bid is equal to current maximum bid
      - $\text{tie} = \text{True}$  # indicate if this is could be a **maximal**
    - \*  $i++$
  - if  $\text{Decrypt}_{\text{FE}}(SK_{\text{Verify}}, \text{Encrypt}_{\text{FE}}(PK_{\text{FE}}, PK_{\text{sign}}^{\text{max}}), c_{\text{bid}}^{\text{max}}, c_{\text{sign}}^{\text{max}}) == 1$   
# We encrypt  $PK_{\text{sign}}^{\text{max}}$  so that it is in the correct form for evaluation using functional encryption
    - \* if  $\text{tie} == \text{True}$ 
      - print “There was a tie, bid again”
    - \* else return  $\text{max}$
  - 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, like another round of bidding.

- 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  $SK_{\text{Verify}}$  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  $m^{\text{max}}$ ) that they are the winner and ask them to send over the value



Then we describe the auction as follows:

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

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

**Bidder Initialization:** Each bidder is expected to send their bid message as an encryption of the bid, an encryption of a signature on that bid, a public key for their signature (since signatures do not guarantee secrecy of what was signed), and the bidder's name. More formally we expect a bidder to:

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

We require the bidder's signature in order 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 circuits be used for keys 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

$$x == y$$

Then descriptions of  $G, E$  and  $\text{Verify}_{\text{sign}}$  are distributed to the bidders and seller, if a participant agrees with the functionality of the 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. This ensures that all participants agree with what information about their bids can be learned from decryption. once these programs are turned into functional encryption secret keys.

**Auction:**

## Table of Contents

<b>Introduction</b>	<b>1</b>
<b>Chapter 1: Background</b>	<b>5</b>
1.1 Encryption	5
1.1.1 Standard Encryption	5
1.1.2 Functional Encryption	7
1.2 Obfuscation	10
1.2.1 Black Box Obfuscation	10
1.2.2 Indistinguishability Obfuscation	11
1.3 Bilinear Maps	12
1.3.1 Definition	12
1.3.2 Intuition	12
1.3.3 Hardness Assumption	13
<b>Chapter 2: Multi-Linear Maps</b>	<b>15</b>
2.1 Cryptographic Multilinear Maps	15
2.1.1 Dream Definition	15
2.1.2 Hardness Assumption	16
2.2 Graded Encoding Schemes	16
2.2.1 Intuition	17
2.2.2 Encodings	17
2.2.3 Definition	18
2.2.4 Procedures	18
2.2.5 Zero Test	19
2.2.6 Hardness Assumptions	20
<b>Chapter 3: Indistinguishability Obfuscation</b>	<b>23</b>
3.1 Circuit Representation	23
3.1.1 Branching Programs	23
3.1.2 Barrington's Theorem	24
3.2 Approaching Obfuscation for $\text{NC}^1$	25
3.2.1 Randomized Branching Programs	25
3.2.2 Kilian's Protocol	25
3.2.3 Encoded Branching Programs	26
3.3 Obfuscation for Poly-sized Circuits	26

3.3.1	Fully Homomorphic Encryption . . . . .	27
3.3.2	Toward Poly-sized Circuit Obfuscation . . . . .	28
3.3.3	Low Depth Non Interactive Zero Knowledge Proofs . . . . .	29
3.3.4	Obfuscation for Poly-Sized Circuits . . . . .	30
<b>Chapter 4:</b>	<b>Functional Encryption . . . . .</b>	<b>31</b>
4.1	Definition . . . . .	31
4.2	Construction . . . . .	32
4.3	Application . . . . .	32
4.3.1	Sealed-Bid First-Price Auction . . . . .	33
4.3.2	Auction Construction . . . . .	33
<b>Chapter 5:</b>	<b>5Gen-C . . . . .</b>	<b>37</b>
5.1	5Gen-C Overview . . . . .	37
5.1.1	libmmap . . . . .	37
5.1.2	CXS . . . . .	37
5.1.3	mio . . . . .	38
5.2	Experiments . . . . .	39
<b>References</b>	<b>. . . . .</b>	<b>41</b>

### 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 one 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 does not 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 *priced*, 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 do not particularly like the art, but do not want to appear rude by bidding extremely low. 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.

### 4.3.2 Auction Construction

A major issue we will need to work around is that because  $i\mathcal{O}$  is currently limited to functions with binary outputs, we can not extract the bid amount from the winning bid’s cipher text without compromising the security of other cipher texts. To work around this we utilize signatures to bind bidders to their encrypted bids, so that the seller can confirm that the winner pays the amount that they bid.

Thus we assume that we have access to a valid and binding digital signature scheme, as described in [Katz & Lindell (2007)], consisting of the polynomial time algorithms:

$$\text{Gen}_{\text{sign}}(\lambda) \rightarrow (PK_{\text{sign}}, SK_{\text{sign}})$$

Generates a public and secret key for signatures

$$\text{Sign}(SK_{\text{sign}}, m) \rightarrow \sigma$$

Takes a secret key and a message and outputs a signature binding the key holder to the message

$$\text{Verify}_{\text{sign}}(PK_{\text{sign}}, m, \sigma) \rightarrow \{0, 1\}$$

Takes a public key and a signature and outputs if the signature and outputs a bid indicating if the signature is valid with respect to the distributor of the public key.

## 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

- $\text{Setup}_{\text{FE}}(\lambda)$ : Takes security parameter  $\lambda$  and generates  
 $(PP, MSK) \leftarrow \text{Setup}_{\text{PKE}}(\lambda)$
- $\text{KeyGen}_{\text{FE}}(MSK, f)$ : Outputs an obfuscation  $SK_f := i\mathcal{O}(P_f(MSK, \cdot))$  for the program  $P_{f, MSK}(c, \pi) =$ 
  - Check  $\pi$  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  $f(\text{Decrypt}_{\text{PKE}}(MSK, c))$
- $\text{Encrypt}_{\text{FE}}(PP, m) \in \{0, 1\}^n$ : Outputs  $c = \text{Encrypt}_{\text{PKE}}(PP, m)$  and  $\pi$  a NIZK proof that  $c$  was produced as just specified.
- $\text{Decrypt}_{\text{FE}}(SK_f, c)$ : 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 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).

As before we will not get into the proof of security, but in [Garg et al. (2013b)] there is a proof that if our NIZK system is computationally zero knowledge, our PKE is CPA-Secure, and our  $i\mathcal{O}$  is indeed an indistinguishability obfuscator; then our functional encryption scheme is indistinguishability secure (as described in section 1.1.2).

## 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 here we construct protocols using functional encryption that are infeasible with current technology and methods of obfuscation.

# Introduction

Historically, encryption has been the cryptographic primitive that allows for two (or more) parties to exchange private information through untrusted channels. That is to say, if Alice wishes to send Bob an invitation to a party, but does not want anyone else to see it, Alice could encrypt the invitation in such a way that only Bob can decrypt it, and then send it over to Bob. If the message is properly encrypted then Alice can send the message through any convenient means without worrying who else might see the message, since *only* Bob can make sense of it through decryption. Indeed we use this type of encryption all the time for sending emails, sms text messages, and even some phone calls so long as we are using the correct apps and settings.

However, in a world where individuals store much of their data on the cloud, the method of encryption just described might be either too limiting or give away too much information. For example, suppose Alice has so many digital photos that she has to store them on Amazing Cloud Services. While Alice likes Amazing Cloud Services, she does not want them to be able to look through all her photos, so she encrypts each photo, along with their affiliated information such as location and date taken, before uploading it to the cloud. Perhaps one day Alice wants to look at all her photos taken in 2008, how should she go about getting these photos from Amazing? Since all the photos are encrypted, Amazing cannot actually look and see which photos were taken when. With the standard notion of encryption, Alice either needs to sacrifice her privacy and teach Amazing Cloud Services how to decrypt her photos so they can send her the ones from 2008, or she would need to download and decrypt each photo in order to determine when it was taken; which would be difficult since her computer space is limited. For situations like this Alice would want to be able to come up with another way to decrypt that gives Amazing the ability to see if an encrypted photo is from 2008 or not, but nothing else. We call encryption schemes that can decrypt to functions of messages rather than just messages, functional encryption schemes.

In order to construct functional encryption, we need another cryptographic primitive known as obfuscation. The goal of obfuscation is to take a program or circuit and make it unintelligible so that running the program on an input reveals nothing else about the input other than the output of the program. Currently, many methods of obfuscation rely on the conversion of circuits or programs into matrices and the encoding of those matrices with a data structure known as a graded encoding scheme that enables computation on encoded inputs so that only information about the final result of the computation can be learned.

In standard encryption (i.e. where we can only get the full message from decrypt-

tion), we expect that the sender and recipient of encrypted messages each have a secret key that allows each of them to encrypt or decrypt messages. Typically these secret keys were random sequences of ones and zeros, however with functional encryption we instead send obfuscated circuits as secret keys. After we are convinced that functional encryption is theoretically possible, we can construct protocols that use it in new and interesting ways. There are also existing frameworks that have instantiated both functional encryption and obfuscation. The 5Gen-C framework allows for us to run experiments on the current efficiency of these primitives and protocols to determine the current feasibility of using them.

Much of the literature that discusses the construction of functional encryption can be dense and unapproachable for an undergraduate computer science student. This thesis was written by and for undergraduates with the intention of being an approachable resource for understanding the basics of functional encryption, obfuscation, how it is constructed, and its applications. This work draws heavily on Garg et al. (2013b) and Garg et al. (2013a); and used software from Galois Inc. as described in Carmer et al. (2017).

## Chapter 4

# Functional Encryption

In this chapter we revisit the definition of functional encryption and then use our understanding of obfuscation to construct it. We then use functional encryption and other cryptographic primitives to describe a protocol for sealed bid first price auctions where only the value of the winning bid can be learned by the auctioneer.

### 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 each function.

**Definition 23.** *Functional Encryption*

A functional encryption scheme for a class of functions  $\mathcal{F} = \mathcal{F}(\lambda)$  over message space  $M \in M_\lambda$  (i.e. messages of length polynomial to  $\lambda$ ) consists of four algorithms  $\mathcal{FE} = \{ \text{Setup}, \text{KeyGen}, \text{Encrypt}, \text{Decrypt} \}$  where:

- $\text{Setup}_{\text{FE}}(1^\lambda) \rightarrow (PP, MSK)$  takes security parameter and outputs Public Parameter and Master Secret Key
- $\text{KeyGen}_{\text{FE}}(MSK, f) \rightarrow SK_f$  takes Master Secret Key and a function  $f \in \mathcal{F}$  and outputs a corresponding secret key  $SK_f$
- $\text{Encrypt}_{\text{FE}}(PP, m) \rightarrow c$  takes Public Parameter and a message  $m \in M$  and outputs cipher text  $c$
- $\text{Decrypt}_{\text{FE}}(SK_f, c) \rightarrow m'$  takes secret key and cipher text and outputs message  $m'$

We say a functional encryption scheme is correct if

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

- 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 that they 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 edit the scheme described in 3.3.2 by 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  $\pi$  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  $\pi$  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  $\pi$  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  $P$ . As a result of this alteration to  $P$ , we expect the evaluator to generate a low depth NIZK proof that their cipher text was generated appropriately.

The only other thing we need to check is that our program  $P \in \text{NC}$  so that we can use  $i\mathcal{O}_{\text{NC}}$  on it. As before we know/assume  $\text{Decrypt} \in \text{NC}$ , and if we use a low-depth NIZK proof system for step 1 then the verifier  $\in \text{NC}$  so  $P \in \text{NC}$ . Thus we have constructed  $i\mathcal{O}_P$  an indistinguishability obfuscater for poly-size circuits. While we will not construct a proof of security, [Garg et al. (2013b)] has a well written hybrid argument that says if our FHE scheme is CPA-Secure and if  $i\mathcal{O}_{\text{NC}}$  is an indistinguishability obfuscater, then the construction above is secure indistinguishability obfuscation for polynomial sized circuits.

## Preliminaries

Although this work is intended to be accessible to undergraduate students, at certain points of this work we will assume that the reader has a basic understanding of modern cryptographic concepts as found in Katz & Lindell (2007). We also assume that the reader is somewhat familiar with group and ring theory as would be taught in an undergraduate abstract or modern algebra course. Although these assumptions are made of the reader, we cite information rather liberally, so even a reader without these prerequisites, with some extra effort, should be able to obtain a reasonable understanding of the topics discussed in this thesis.

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

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  $\text{Obfuscate}(C_0) \rightarrow PP_0$  and  $\text{Obfuscate}(C_1) \rightarrow PP_1$  and give  $PP_b$  to the adversary  $\mathcal{A}$  for  $b \leftarrow \{0, 1\}$  chosen uniformly, consider the following attack.

Upon receiving  $PP_b = (\mathcal{P}_b, PK_{\text{FHE}}^b, g_b)$ ,  $\mathcal{A}$  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  $\ell$  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  $w = 11 \dots 100 \dots 0$ . Then  $\mathcal{A}$  encrypts the mask  $w_{\text{Enc}} = \text{Encrypt}_{\text{FHE}}(PK_{\text{FHE}}^b, w)$  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  $\mathcal{P}_b(\text{Eval}_{\text{FHE}}(PK_{\text{FHE}}, U_\lambda(\cdot, m), g'_b) \rightarrow b'$ .

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 Victor be our verifier. Given a statement and a witness to that statement a Peggy should be able to generate a proof  $\pi$  such that when  $\pi$  is given to Victor, he is convinced that  $\pi$  is a valid proof and yet anyone given  $\pi$  should be unable to use  $\pi$  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  $\pi$  from the prover to the verifier where  $\pi$  has the following properties:

- Perfect Completeness: A proof system is perfectly complete if an honest prover can generate a proof  $\pi$  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

to see intermediate 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 ( $\text{Dec}_{\text{FHE}}(sk, \text{Eval}_{\text{FHE}}(pk, \text{average}, c_1, \dots, c_\ell))$ ), but also decrypt any sub result of the computation he wants ( $\text{Dec}_{\text{FHE}}(sk, c)$  for any  $c$ ) including the raw data itself thus compromising the security of Alice's data. As we discussed in section 1.1.2 this would be a great situation for using functional encryption.

### 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  $\text{NC}^1$  which we will denote  $i\mathcal{O}_{\text{NC}}$ , and a fully homomorphic encryption scheme ( $\text{Setup}_{\text{FHE}}, \text{Encrypt}_{\text{FHE}}, \text{Eval}_{\text{FHE}}, \text{Decrypt}_{\text{FHE}}$ ) where we can assume  $\text{Decrypt}_{\text{FHE}}$  can be realized by a family of circuits in  $\text{NC}^1$  [Gancher (2016)]. 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  $\lambda$  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  $U_\lambda$  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  $\lambda$ .

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 since we can functionally encrypt circuits of polynomial size and run evaluation with  $g$ , and the circuit encrypted to get  $g$  should be “well hidden” since we assumed our FHE scheme to be correct and secure. 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

## 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 Standard 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 use random secret keys to conceal the message. 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  $K$ : “the key space”; the set of all valid messages  $M$ : “the message space”, and the set of all valid cipher texts (encrypted messages)  $C$ : “the cipher space”.

**Definition 1** (Standard Encryption Scheme).

$$\text{Gen} : \mathbb{Z} \rightarrow K \times K$$

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

$$\text{Enc} : K \times M \rightarrow C$$

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

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

Defined for key  $sk \in K$  and cipher text  $c \in C$ ; to be  $Dec_{sk}(c) \rightarrow m$  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  $\Pi$  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 wants to send Bob a secret message  $m$ . To do this in the public key setting Bob would have to run  $Gen(n) \rightarrow (pk, sk)$  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.

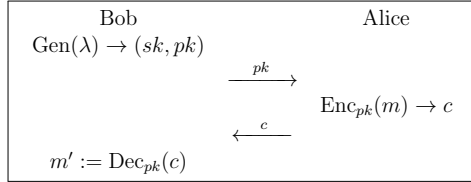


Table 1.1: Public Key Encryption Protocol

We also care about the security of an encryption scheme. That is to say, we want to ensure that an encryption scheme is *good* at hiding the information that we are encrypting. There are many different types of security, here we will focus on CPA-Security. Informally, we say an encryption scheme is secure against chosen plaintext attacks (hereby CPA-Secure) if an adversary is unable to differentiate the cipher texts from two known and distinct messages.

More formally, we define CPA security with the following game:

**Definition 3.** CPA-Game and Security

Suppose we have the following encryption scheme  $(Gen, Enc, Dec)$  with  $(pk, sk) \leftarrow Gen(\lambda)$ . Then for any adversary  $\mathcal{A}$ , we give them  $pk$  and our encryption function  $Enc$ . They can then encrypt whatever messages they want, and eventually give us the

### 3.3.1 Fully Homomorphic Encryption

We will not get into much detail as to how FHE is constructed, instead we will describe what is expected of an 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 22.** 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:

- $KeyGen_{FHE} : \mathbb{Z} \rightarrow K \times K$   
Defined  $KeyGen_{FHE}(\lambda) \rightarrow (pk, sk)$  where  $\lambda$  is the security parameter and  $pk, sk$  are a public and secret key respectively
- $Enc_{FHE} : K \times \mathcal{M} \rightarrow C$   
Defined  $Enc_{FHE}(pk, m) \rightarrow c$  meaning anyone with the public key can encrypt a message and get back a cipher text
- $Dec_{FHE} : K \times C \rightarrow \mathcal{M}$   
Defined  $Dec_{FHE}(sk, c) \rightarrow m$  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  $\lambda$  and not the input.
- $Eval_{FHE} : K \times \mathcal{F} \times C \times C \times \dots \times C \rightarrow C$   
Defined  $Eval_{FHE}(pk, f, c_1, c_2, \dots, c_\ell) = c_f$  for  $f \in \mathcal{F}$ .

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

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

As opposed to the notion of functional encryption introduced in the first chapter, with FHE decryption is all or nothing just like with standard encryption. A good use for FHE would be if Alice wants to have a large amount of data on Bob's cloud storage, an untrusted server, and wants to be able to run computations on this stored data remotely. If Alice encrypts her data using a FHE scheme, she can then as Bob to run Eval commands on the data and then send it over to her, when she can then download the output of the evaluation command, and decrypt to find its value. This works well because Alice does not 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 an intermediate 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



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

While this is a good start toward obfuscation and may in fact seem like all we need to obfuscate a circuit, we need guarantees that intermediate results are not revealed through the programs execution.

### 3.2.3 Encoded Branching Programs

In order to get some of the guarantees we are looking for with circuit obfuscation, we use 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 their 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.

At a first, encoded branching programs might seem like all we need for circuit obfuscation. After all, the hardness assumptions we defined in section 2.2.6, let us say that computation done with encodings from a graded encoding scheme guarantees that the multiplication and addition operations in matrix multiplication will not give away information about intermediate results of the computation. However encoded branching programs do not 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  $A_{i,\text{inp}(i)}$  inconsistently in the sense that the  $j$ th bit of the input might be treated as 1 the first time  $\text{inp}(i) = j$  and then 0 the next time  $\text{inp}(i) = j$ . 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  $i\mathcal{O}_{\text{NC}}$ , are highly technical and do not add much to understanding  $i\mathcal{O}_{\text{NC}}$ . 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 NC<sup>1</sup> along with Fully Homomorphic Encryption (FHE) to get obfuscation for polynomial sized circuits.

messages  $m_0$  and  $m_1$ . We then run  $\text{Enc}(pk, m_b) \rightarrow c_b$  for  $b$  chosen with probability  $1/2$ , and give  $c_b \rightarrow \mathcal{A}$ . After some time  $\mathcal{A}$  outputs a bit  $b'$ .

We say  $\mathcal{A}$  is successful if  $b = b'$  and we say an encryption scheme is CPA-Secure if no adversary is able to succeed at the CPA-Game with probability more than  $1/2$ .

### 1.1.2 Functional Encryption

With standard encryption, decryption is all or nothing, either you have the secret key and can find out the message, or you do not have the secret key so you can not learn anything about the encrypted message. With functional encryption we broaden the possibilities of what is communicated between senders in an encryption scheme. We start with a definition and then formalize the scheme.

**Definition 4** (Correctness). A Functional Encryption Scheme  $\Pi$  is **correct** if for  $m \in M$ , some predetermined function  $f$  with  $M$  as its domain, and appropriate public key and evaluation key  $(PK, SK_f) \in K$  from  $\Pi$ 's key generation algorithm:

$$\text{Dec}(SK_f, \text{Enc}_{PK}(m)) = f(m)$$

It is 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 5** (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  $\Pi$  is defined to be the following four algorithms:

$$\text{Setup} : \mathbb{Z} \rightarrow K \times K$$

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

$$\text{Gen} : K \times \mathcal{F} \rightarrow K$$

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

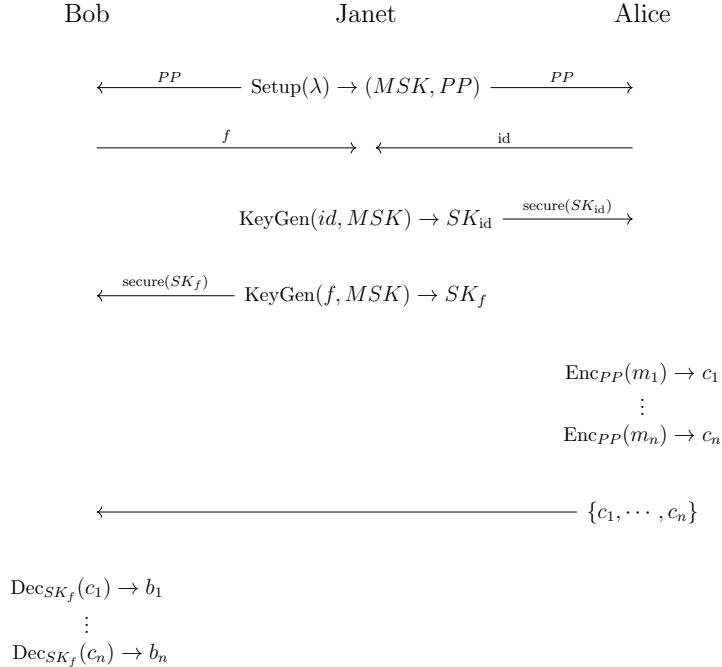
$$\text{Enc} : K \times M \rightarrow C$$

Defined: For  $PP \in K$  and  $m \in M$ ; to be  $\text{Enc}_{PP}(m) \rightarrow c$

$$\text{Dec} : K \times C \rightarrow M$$

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

Suppose Alice wants to store some text files ( $\{m_1, m_2, \dots, m_n\}$ ) 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 does not 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  $\text{Setup}(\lambda) \rightarrow (PP, MSK)$  and publishes the public parameter, and also runs  $\text{Keygen}(MSK, id) \rightarrow SK_{id}$  and  $\text{Keygen}(MSK, f) \rightarrow SK_f$ . Then Janet securely sends  $SK_{id}$  to Alice and  $SK_f$  to Bob. Now Alice encrypts her text files  $\text{Enc}_{PP}(m_1) \rightarrow c_1, \text{Enc}_{PP}(m_2) \rightarrow c_2, \dots, \text{Enc}_{PP}(m_n) \rightarrow c_n$ , and sends  $\{c_1, c_2, \dots, c_n\}$  to Bob for storage. Bob then runs  $\text{Dec}_{SK_f}(c_1) \rightarrow b_1, \dots, \text{Dec}_{SK_f}(c_n) \rightarrow b_n$  and allows storage of all  $c_i$  that indicate the word "cat" is not being stored. This interaction is illustrated in the following diagram:



Since Alice has  $SK_{id}$  if she want to access a poem that she had written and stored on Bob's server, she can request that Bob send over her data, at which point she can run  $\text{Dec}_{SK_{id}}$  on the cipher text containing her poem so that she can read it in full. Of course having to download all the data she had stored is not optimal, and neither is having to decrypt all the cipher texts to find the poem she is looking for, but this is

### 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  $C$ .

Thus we now know that circuits can be represented as matrix branching programs efficiently, so long as we keep the circuits of log depth (i.e. in  $\text{NC}^1$ , the complexity class of log time in polynomial parallel processors)

## 3.2 Approaching Obfuscation for $\text{NC}^1$

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  $\text{NC}^1$  hereby referred to as  $i\mathcal{O}_{\text{NC}}$ .

### 3.2.1 Randomized Branching Programs

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

#### Definition 21. Randomized Branching Programs

Given a branching program  $\mathcal{BP} = (\text{inp}(i), A_{i,0}, A_{i,1})_{i=1}^n$  and  $n$  random  $5 \times 5$  invertible matrices  $R_1, R_2, \dots, R_n$ . 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

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

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 is not commutative, yet for  $A, B, C \in \{0, 1\}^{5 \times 5}$

$$\tilde{A}\tilde{B}\tilde{C} = (AR_1)(R_1^{-1}BR_2)(R_2^{-1}C) = AIBIC = ABC$$

### 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 obfuscator 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  $m$ . (for more detail see [Kilian (1988)])

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  $\tilde{A}_{i, \text{inp}(i)}$  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  $y$ .

bers  $x_1x_2 > x_3x_4$  is given by:

$i$	$\text{inp}(i)$	if $\text{inp}(i) = 0$ go to	if $\text{inp}(i) = 1$ go to
1.	1	2	3
2.	3	4	$\theta$
3.	3	$I$	4
4.	2	$\theta$	5
5.	4	$I$	$\theta$

In the above program, the first column  $i$  indicates the current line of the program we are at. The second column  $\text{inp}(i)$  indicates the index of the input to focus on (i.e.  $\text{inp}(1)$  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  $\text{inp}(i)$ ; or an output value  $\theta$  or  $I$ . To work through an example, consider checking if  $10 > 11$  with the above program where  $x_1 = 1, x_2 = 0, x_3 = 1$ , and  $x_4 = 1$ . We start at  $i = 1$  and check  $x_1$ , since  $x_1 = 1$  we are directed to  $i = 3$  where we check  $x_3$ , since  $x_3 = 1$  we then jump to  $i = 4$  where we examine  $x_2$ , since  $x_2 = 0$  we are instructed to output 0.

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

**Definition 20.** (*Oblivious*) *Matrix Branching Program*

An oblivious branching program of length- $n$  for  $\ell$ -bit inputs is a sequence

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

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

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

Where in the example the program chose one of two line numbers to jump to depending on the value of the bit at  $\text{inp}(i)$ , 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. While the proof of the Barrington's Theorem is rather involved and technical, it uses the recursive structure of boolean circuits to inductively show how "OR", and "NOT" gates (and since NOR is a universal gate this describes all circuits) can be simulated in the  $5 \times 5$  matrix representation of elements in the symmetric group [see: Barrington (1986)].

just a toy example for us to get a taste for using functional encryption. We will get into more detailed protocols in chapter 4.

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 she 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.

As with the standard notion of encryption, we want to be able to discuss the security of a functional encryption scheme. While we would like a functional encryption scheme to be CPA secure against eavesdroppers who do not have secret keys, we also need to be sure that parties with secret keys are unable to learn more about cipher texts than what they learn just from the decryption function. As before our security definition is game based, although this time we call it indistinguishability security.

**Definition 6.** *Indistinguishability Security*

Let  $(\text{Setup}, \text{Gen}, \text{Enc}, \text{Dec})$  be a functional encryption scheme and  $(PP, MSK) \leftarrow \text{Setup}(\lambda)$ . Then for an adversary  $\mathcal{A}$

- We give  $PP$ ,  $\text{Enc}$ , and  $\text{Dec}$  to  $\mathcal{A}$  who can then query for a secret key that computes  $f_i$ . After receiving a query for  $f_i$  we run  $\text{Gen}(MSK, f_i) \rightarrow SK_{f_i}$  and send  $SK_{f_i}$  to  $\mathcal{A}$ .
- Eventually  $\mathcal{A}$  sends us two messages  $m_0, m_1$  where  $f_i(m_0) = f_i(m_1)$  for all functions queried so far. We then run  $\text{Dec}(MSK, m_b) \rightarrow c_b$  for  $b$  chosen with probability  $1/2$  and send  $c_b$  to  $\mathcal{A}$ .
- $\mathcal{A}$  can continue to send us queries for functions  $f_i$  where  $f_i(m_0) = f_i(m_1)$  which we will generate and send keys for. After some time  $\mathcal{A}$  outputs a bit  $b'$ .

As before we say  $\mathcal{A}$  is successful if  $b = b'$ , and we say a functional encryption scheme is indistinguishability secure if no adversary is able to succeed with probability greater than  $1/2$ .

Simply put, our security definition says that no adversary should be able to use a secret key  $SK_f$  to learn anything else about a cipher text than the output of  $f$  on the encrypted message. We require that  $f(m_1) = f(m_0)$  for all queries because we want individuals with secret keys to be able to functionally decrypt, but this would differentiate any two messages where  $f(m_0) \neq f(m_1)$ . This security definition encapsulates CPA security since if an adversary can distinguish two messages just from their cipher text then they would be able to consistently succeed at the indistinguishability game. We will not be doing any security proofs in this work, so this is about the extent to which we will cover security definitions of functional encryption, but for further discussion on security definitions see [Boneh et al. (2011)].

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 functionalities 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 work with later on.

## 1.2 Obfuscation

A concern that needs to be addressed in order for us to construct functional encryption is: how can we ensure that some adversary given a secret key  $SK_f$  for functionality  $f$  cannot alter  $SK_f$  after it has been generated to reveal more information about cipher texts than intended?

For example, suppose  $f(x_1, x_2) = x_1 + x_2$ , since cryptographer's assume Kerckoff's principle that the way  $PK_f$  is generated should be public, an adversary might be able to take advantage of the ordering of the sum in  $f$  in order to learn the value of  $x_1$  and  $x_2$ .

To prevent the functionality of  $SK_f$  from being altered, we want to obfuscate  $f$  so that no adversary can take advantage of how  $SK_f$  is generated from  $f$  to alter the key's functionality. It is easy to obfuscate functions, programs, encodings, etc... but it is difficult to *provably* obfuscate anything, after all defining obfuscation is not easy to conceptualize. In this section we discuss different attempts at formalizing obfuscation and in chapter 3 we will construct it.

### 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. The goal of black box obfuscation as defined in [Barak et al. (2001)] is that a circuit that has been black box obfuscated should reveal no more about its inner workings than a table of inputs and outputs for the same circuit (we call this table an Oracle).

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 it would be possible to obfuscate a private key encryption function with a secret key  $sk$  baked in  $\mathcal{O}(\text{Enc}_{sk}(\cdot)) \rightarrow \text{Enc}(\cdot)$  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.

## Chapter 3

## Indistinguishability Obfuscation

In this chapter we use the result of Barrington's theorem [Barrington (1986)] to describe small circuits in such a way that they can be encoded under a graded encoding scheme and randomized to yield circuit obfuscation for  $\text{NC}^1$ . Then we use fully homomorphic encryption (section 3.3.1) and non interactive zero knowledge proofs (section 3.3.3) to broaden indistinguishability obfuscation for  $\text{NC}^1$  to obfuscation for polynomial sized circuits.

### 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  $(x \text{ and } y) \text{ or } z$ . 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 how  $C$  computes on inputs.

#### 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  $\theta$  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 binary num-

### 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 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 obfuscator  $i\mathcal{O}(\cdot)$ ; we say that  $i\mathcal{O}$  is an **indistinguishability obfuscator** if for all Distinguishers  $\mathcal{D}$

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

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 obfuscator.

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 watermarks 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 does not change the functionality of his copy of the program, but indicates that it 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 obfuscator before uploading it, 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, in [Goldwasser & Rothblum (2007)]. As a result 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.

In cryptography, a common way to hide information is by randomization, yet to implement obfuscation we need a way of randomizing computations so that they

cannot be reverse engineered since the order of computation could be utilized by a distinguisher to differentiate two circuits with the same functionality. To do this we utilize a data structure called a graded encoding scheme which relies heavily on multilinear maps.

### 1.3 Bilinear Maps

Lastly we give a brief introduction to bilinear maps, and the properties they have which we will generalize to multilinear maps in chapter 2.

#### 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  $e : G_1 \times G_2 \rightarrow G_3$  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 in the clear without revealing their value.

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  $(e \in \mathbf{params}, g_1, pk_a := g_1^a)$ . 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 bilinear map makes it possible for 3 parties to share a secret without risk of an eavesdropper also being able to know the secret.

**Definition 18.** *Graded DDH*

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

Given  $\mathcal{S}, u_1, u_2, \dots, u_{k+1}$ , and  $\mathbf{u}$ , determine if  $\mathbf{u}$  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.

**Definition 19.** *Graded Discrete Log*

Given  $a \in S_i^{(\alpha)}$  for  $\alpha \in R_q$  and  $0 < i$ , obtain a level-zero encoding of  $\alpha$ .

We will not go much into how *good* these assumption are, but we will say that this encoding scheme is far less studied than, say, RSA and so the cryptanalysis is much less extensive, so it is vulnerable to new attacks. 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. The important thing to be weary of when using graded encoding schemes is just that it will be a while longer before all the “easy” attacks have been found. While this insecurity is suboptimal for many applications, we will say that graded encoding schemes are good enough for our purposes.

As before we will not 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

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

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, but it is rather similar in form to DDH and discrete log.

### 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 9.** *Bilinear Computational Diffie-Hellman Problem*

Let  $G_1, G_2$  be cyclic groups of prime order  $p$  and  $e : G_1 \times G_1 \rightarrow G_2$  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 10.** *Bilinear Discrete Logarithm* 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  $a \in \mathbb{Z}_p$  chosen uniformly, and generators  $g_1, g_2$  of  $G_1, G_2$  respectively, the bilinear discrete logarithm problem is: given  $g_1, g_1^a, G_1, G_2$ , and  $e$  find  $\alpha$ .

Note that while the above problem may seem easy, the encoding  $g_1^a = g$  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.

- Outputs:  $\mathcal{S}$ , the description of a  $k$ -graded encoding scheme as described above, the principle prime ideal  $I$ , and  $p_{zt}$  a zero-test parameter which we will explain with some detail in section 2.2.5.
- **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 (see Garg et al. (2013a)), 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.
- **Encode:** 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 have already described. Note that whoever runs the **Instance Generation** know  $I$  so they can encode any plaintext they want

So we are able to encode elements, as well as do computation on them. The only procedure we have yet to cover is one that allows us to *learn* from our computation. This procedure is rather involved, so we cover it in depth in section 2.2.5.

### 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 whether 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 17.** *Zero Test*

Let  $R$  be a cyclotomic ring,  $R_q = R/qR$  for prime  $q \in R$ , small  $g, h \in R_q$ ,  $z \in R_q$  chosen uniformly (not 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.



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

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

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

Thus this encoding gives us the ability to do operations (addition and multiplication) we were looking for.

### 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 16.** *k-Graded Encoding Scheme*

A *k-Graded Encoding System* consists of a ring  $R$  and a system of sets

$$\mathcal{S} = \{S_i^{(\alpha)} = \alpha + I \mid \forall \alpha \in R, 0 \leq i \leq k\}$$

, such that:

1. For every fixed index  $i$ , the sets  $\{S_i^{(\alpha)} \mid \alpha \in R\}$  are disjoint (meaning they form a partition of  $S_i := \cup_{\alpha} S_i^{(\alpha)}$ )
2. There is an associative binary operation  $'+'$  and a self-inverse unary operation  $'-'$  such that  $\forall \alpha_1, \alpha_2 \in R$ , index  $i \leq k$ , 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:  $\lambda$ , our security parameter, and  $k$  the maximum number of multiplication operations our computation might need (note how this will be the degree of our  $k$ -graded encoding scheme)

## Chapter 2

## Multi-Linear Maps

In this chapter we use what we learned about bilinear maps in section 1.3 to give a generalized definition of multilinear maps. Then we use bilinear maps to describe a simplified definition of graded encoding schemes that allow for addition, multiplication, negation, and a new operation called the zero test. Once we are comfortable with the simplified definition of graded encoding schemes we specify how we will need to encode inputs in order to get the desired properties of the simplified definition, and the formal definition will follow.

### 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; as well as learn about the final result of the computation.

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 11.** *Cryptographic Multilinear Map*

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

1. For  $g_i \in G_i \forall i \in [K]$  where  $[K]$  is the set  $\{1, 2, \dots, k\}$ , and  $\alpha \in \mathbb{Z}_p$  then  $e(g_1, g_2, \dots, g_k^\alpha, \dots, g_k) = e(g_1, g_2, \dots, g_k, \dots, g_k)^\alpha$
2. if  $g_i$  generates  $G_i$  then  $e(g_1, g_2, \dots, g_k)$  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  $g_i^\alpha$  as a public encoding of secret  $\alpha$ . 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  $e : G_1 \times G_2 \times G_3 \rightarrow G_4$  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 decisional Diffie-Hellman problem and discrete logarithm problem.

**Definition 12.** *Multilinear Decisional Diffie-Hellman Problem*

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

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

**Definition 13.** *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.

We assume both of these problems to be hard.

## 2.2 Graded Encoding Schemes

As we have seen,  $k$ -linear maps allow us to multiply up to  $k$  encoded secrets together in the clear while keeping the secrets private. Now we will build up an understanding as to *how* we can encode elements in such a way that allows for not just multiplication in the form of multilinear maps, but addition as well. Not only do we want to be able to multiply and add secrets, but we would also like to somehow *extract* information about the final output of the computation, after all what good is doing a computation if nothing can be learned from it. We will address what graded encoding schemes are and how they meet all of our needs in this section.

### 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 while building up to the actual definition.

**Definition 14.** *k-Graded Encoding Scheme (simplified)*

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

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

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

In this Encoding Scheme, secrets  $\alpha_1, \alpha_2, \dots \in \mathbb{Z}_p$  are initially encoded as they were with bilinear maps, by taking a generator  $g_1 \in G_1$  and raising it to the  $\alpha_1$  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  $\alpha_3$  we can ‘multiply’ the level one encoding  $\alpha_3$  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  $e_{2,2}(g_2^{\alpha_3}, g_2^{\alpha_1 \cdot \alpha_2}) = g_4^{\alpha_3 \cdot \alpha_1 \cdot \alpha_2}$  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 Definition 12 is a simplified definition of graded encoding schemes that gives an intuition into the *graded* structure of the scheme. In our simplified definition, encodings are unique and of the form  $g_i^\alpha$ . 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  $\alpha$  at level  $i$  :  $S_i^{(\alpha)}$ .

**Definition 15.** *Level  $i$  Encoding*

Let  $R$  be a cyclotomic ring (i.e. a ring of the form  $\mathbb{Z}[X]/(X^n + 1)$ ),  $R_q = R/qR$  for large prime  $q \in R$ ,  $g \in R_q$  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  $\alpha \in R_q \cong \mathbb{Z}_p$  to be anything in the set

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

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