UCLA

UCLA Electronic Theses and Dissertations

Title

Indistinguishability Obfuscation from Well-Studied Assumptions

Permalink

https://escholarship.org/uc/item/7vq3z6v1

Author

Jain, Aayush

Publication Date

2021

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

UNIVERSITY OF CALIFORNIA

Los Angeles

Indistinguishability Obfuscation from Well-Studied Assumptions

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Computer Science

by

Aayush Jain

2021

© Copyright by

Aayush Jain

2021

ABSTRACT OF THE DISSERTATION

Indistinguishability Obfuscation from Well-Studied Assumptions

by

Aayush Jain
Doctor of Philosophy in Computer Science
University of California, Los Angeles, 2021
Professor Amit Sahai, Chair

Can we *efficiently* compile a computer program P into another one say $\tilde{P}$, which has the same functionality as P on every input, but otherwise is as *unintelligible* as possible? This question was asked in 1976 by the Turing award winning work of Diffie-Hellman (STOC 1976). Such a compiler is called as Obfuscation. It is not hard to see that such a notion will find lots of application so much so that today it is one of the most versatile primitives in cryptography. The work of Barak et. al. (Crypto 2001) and Goldwasser and Rothblum (TCC 2007) established that in order to construct an obfuscation scheme which provides best possible unintelligibility guarantees, it suffices to construct what is called as an Indistinguishibility Obfuscation ($i\mathcal{O}$) scheme. An $i\mathcal{O}$ scheme is an obfuscation scheme that only hides implementation differences. Such a scheme guarantees that for every two programs P_0, P_1 of the same description size, same running time having the same input output behavior, but otherwise having different internal details $i\mathcal{O}(P_0)$ is indistinguishabile to $i\mathcal{O}(P_1)$. Unfortunately, up until now it was not clear if such a primitive can even exist. All the known constructions were heuristic, based on newly stated assumptions, and most of the times subject to cycles of attacks and fixes. In this thesis, we present the first feasibility result of an obfuscation scheme by constructing an obfuscation scheme from well-studied cryptographic assumptions. We prove:

Theorem: (Jain-Lin-Sahai STOC 2021, Jain-Lin-Sahai 2021b) *Assume sub-exponential security of the following assumptions:*

- *the Learning Parity with Noise (*LPN*) assumption over general prime fields* $\mathbb{Z}_p$ *with polynomially many* LPN *samples and error rate* $1/k^{\delta}$*, where* k *is the dimension of the* LPN *secret, and* $\delta > 0$ *is any constant;*
- *the existence of a Boolean Pseudo-Random Generator (*PRG*) in* NC^0 *with stretch* $n^{1+\tau}$*, where* n *is the length of the* PRG *seed, and* $\tau > 0$ *is any constant;*
- *the Decision Linear (*DLIN*) assumption on symmetric bilinear groups of prime order.*

Then, (subexponentially secure) indistinguishability obfuscation for all polynomial-size circuits exists.

All the three assumptions are based on computational problems with a long history of study, rooted in complexity, coding, and number theory. As a corollary this gives feasibility results for many of the other cryptographic primitives that had remained elusive and are known to follow from $i\mathcal{O}$, from the same set of assumptions. Our work also gives rise to the first construction of a fully-homomorphic encryption scheme that does not rely on the hardness of a lattice based problem.

This work is a culmination of a line of work spanning multiple years [AJS18, AJL$^+$19, JLMS19, BHJ$^+$19, GJLS21, JLS21a, JLS21b]. This line introduced many different intermediate building blocks relying on a number of concepts in cryptography. These blocks were simplified and built upon in subsequent works. In this thesis, we present a self contained, simplified and streamlined construction largely adapted from [JLS21b] but borrowing elements from some of the works mentioned above.

The dissertation of Aayush Jain is approved.

Eli Gafni

Alexander Sherstov

Tony Nowatzki

Amit Sahai, Committee Chair

University of California, Los Angeles

2021

“To my wife and my family”

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

First and foremost, I am extremely grateful to my advisor Amit Sahai. Since the very beginning, we hit it off really well. He brought the best out of me by helping to identify my strengths, weaknesses and research tastes and gradually shaping me as a researcher. He would identify exactly what I need at right points in time, be it frequent meetings or long intervals without one or sometimes just a morale boost. He has always encouraged me to be (technically) fearless, for which I can't thank him enough. I will miss our mid-night meetings and candid talks on the fifth floor.

Vipul Goyal and Rachel Lin have also played an extremely important role in my development as a researcher. I met with Vipul during my undergrad studies, and he mentored me for over two years. I learned many technical and non technical aspects about research from him. Rachel has been an extremely close collaborator in all the works in this thesis. Our collaboration started in 2018, and has continued since. I am always amazed at how energetic she is in research and this is something that inspires me. I hope to develop many more collaborations with her in the future. Early on in my career, Ankit Gupta and Prabhanjan Ananth set good examples for me as senior graduate students. I am thankful to Ankit for giving me brutally honest advice on several matters related to research styles. Prabhanjan and I had countless bus rides, sunday runs, which were made so much fun by discussing research questions.

Yuval Ishai and Boaz Barak are extremely knowledgable; talking to them has always been really illuminating and this has always have an impact on my choice of research problems. I am very thankful to Tarun Kathuria, Sam Hopkins, Pravesh Kothari and Prasad Raghavendra for answering my countless oracle calls about related questions in algorithms. Tal Rabin spent hours with me, giving me feedback on my presentations. Her feedback really helped me a great deal in improving my presentation skills.

I am thankful to my wonderful UCLA professors, in particular, Sasha Sherstov, Raghu

Meka, Rafail Ostrovsky, Guy Van Den Broeck, Todd Millstein, Eli Gafni and Tony Nowatzki. I had countless illuminating discussions with them which helped me gain a wider perspective about different research areas in computer science. At UCLA, Osanna Kazarian, Joseph Brown, Freda Robinson, Therese Garcia and Mildri Lopez-Duarte made sure that I never ran into administrative problems.

Saikrishna Badrinarayanan, Dakshita Khurana, Ilan Komargodski and Adam O'Neill are great personal friends and fun collaborators. I am very thankful to Rex Fernando, Nathan Manohar, Alexis Korb, Paul Lou, Pei Wu, Ashutosh Kumar, Eric Miles, Alain Passelégue, Peter Rassmussen, and Divya Gupta for being such wonderful friends and fun lab mates and making my experience at UCLA memorable. Working in a tightly knit community I was very fortunate to make really great personal friends and collaborators in Akshayaram Srinivasan, Romain Gay, Mohammad Hajiabadi, Rishab Goyal, Venkata Kopulla, Fermi Ma, James Bartusek, Ji Luo, Willy Quach, Sam Kim, David Wu, Alex Lombardi and others I am sure I missed writing about.

I am extremely grateful to Tatsuaki Okamoto and Ilan Komargodski at NTT for their support and encouragement and for the financial support I received from Google PhD fellowship.

A special thanks goes to Gaurav Agarwal and Hemant Saggar for being caring roommates who supported me in every situation throughout this journey. I am also thankful to friends at UCLA: Saswat Padhi, Mohammad Ali Gulzar, Kirti Dhwaj, Jaskaran Singh Grover, Aishwarya Sivaraman, Murali Ramanujan, Siva Kesava, Pradeep Dogga, Zeina Migeed, Vidushi Dadu, Arjun Akula, Eli Gaffe, Shuyang Liu and Akshay Utture, for all the fun times.

Most importantly, I am thankful to my wife Eleena, my parents and my family for keeping my spirits high, and their unconditional love and support during this long journey.

VITA

2013	B.Tech in Electrical Engineering Indian Institute of Technology, New Delhi
2013	M.Tech in Electrical Engineering Indian Institute of Technology, New Delhi
2013-2015	Research Fellow Microsoft Research, India
2016	Dean's Fellowship, UCLA
2018	Google PhD Fellowship
2020	Symantec Outstanding Graduate Student Award, UCLA

SELECTED PUBLICATIONS

Samuel B. Hopkins, Aayush Jain and Huijia Lin. Counterexamples to New Circular Security Assumptions Underlying iO. CRYPTO 2021.

Romain Gay, Aayush Jain, Huijia Lin and Amit Sahai. Indistinguishability Obfuscation from Simple-to-State Hard Problems: New Assumptions, New Techniques, and Simplification. EUROCRYPT 2021.

Fabrice Benhamouda, Aayush Jain, Ilan Komargodski and Huijia Lin. Multiparty Reusable Non-interactive Secure Computation from LWE. EUROCRYPT 2021.

Aayush Jain, Huijia Lin and Amit Sahai. Counterexamples to New Circular Security Assumptions Underlying iO. STOC 2021. Co-winner of the Best Paper Award.

Saikrishna Badrinarayanan, Rex Fernando, Aayush Jain, Dakshita Khurana and Amit Sahai. Statistical ZAP Arguments. EUROCRYPT 2020.

James Bartusek, Yuval Ishai, Aayush Jain, Fermi Ma, Amit Sahai and Mark Zhandry. Affine Determinant Programs: A Framework for Obfuscation and Witness Encryption. ITCS 2020.

Prabhanjan Ananth, Aayush Jain, Huijia Lin, Christian Matt and Amit Sahai. Indistinguishability Obfuscation Without Multilinear Maps: New Paradigms via Low Degree Weak Pseudorandomness and Security Amplification. CRYPTO 2019.

Boaz Barak, Samuel B. Hopkins, Aayush Jain, Pravesh Kothari, and Amit Sahai. Sum-of-Squares Meets Program Obfuscation, Revisited. EUROCRYPT 2019.

Aayush Jain, Huijia Lin, Christian Matt and Amit Sahai. How to Leverage Hardness of Constant-Degree Expanding Polynomials over $\mathbb{R}$ to build iO. EUROCRYPT 2019.

Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter M. R. Rasmussen and Amit Sahai. Threshold Cryptosystems from Threshold Fully Homomorphic Encryption. CRYPTO 2018.

Prabhanjan Ananth, Aayush Jain, Moni Naor, Amit Sahai, Eylon Yogev. Universal Constructions and Robust Combiners for Indistinguishability Obfuscation and Witness Encryption. CRYPTO 2016.

CHAPTER 1

Introduction

In this thesis, we consider an extremely fundamental question:

Is it possible to efficiently turn a computer program into an unintelligible version of itself?

To make the question more precise, we want to build an efficient (polynomial time) compiler $\mathcal{O}$ that takes as input a program P (for simplicity, say represented as a Boolean circuit) and outputs another circuit $\mathcal{O}(P) = \tilde{P}$. We require that this program has identical functionality to the original program P, that is on every input x, $P(x) = \tilde{P}(x)$. The size of $\tilde{P}$ is allowed to be polynomially larger than the size of P. The main punchline however is the security requirement: the program $\tilde{P}$ should hide as much information as possible about the implementation details of P, thereby making it as unintelligible as possible. We will call $\mathcal{O}$ an obfuscator and $\tilde{P}$ as an obfuscation of P.

We would also like to stress that we require $\tilde{P}$ to be in the *same format* as P. That is, $\tilde{P}$ is described as a Boolean circuit. We stress that, therefore, the implementation of $\tilde{P}$ is completely public. In other words, there is no secure hardware to assist us with the task. Indeed, if we assumed that secure hardware existed, then an intuitive idea is to use secure hardware to solve this problem by wrapping the circuit P inside a secure hardware token. However, such a token can be evaluated only by the person who has physical access to the token. In this, there is also an implicit assumption that the token resists all side channel attacks which is a strong assumption. On the other hand, we are asking for a fully software solution. In particular, $\tilde{P}$ can be published on a bulletin board for everyone to see and then used by anyone who wants to use it.

We are going to delay defining the formal details of what it means to be "as unintelligible as possible" (we will address this in Section 1.1). Until then, let us discuss why such a notion might be useful. Indeed it is not hard to come up with applications of such a primitive. In fact, when this question was posed for the first time by the Turing award-winning work of Diffie-Hellman [DH76], the motivating application was to construct a *public-key encryption* scheme from a secret-key encryption scheme. Back then, public-key encryption was not known to exist and in their quest for building a public-key encryption scheme Diffie and Hellman suggested the following high level approach: First choose the secret key K to be a randomly chosen secret key for a secret-key encryption scheme. To create the public key, use $\mathcal{O}$ to obfuscate a Boolean circuit G that takes as input a message m and a randomness r, derives randomness appropriately from the message m and r, and then computes and outputs an encryption $\mathsf{CT} = \mathsf{Enc}(K, m)$ using the derived randomness, where the secret key K is hardwired into G. Let us call the obfuscated circuit $\tilde{G}$.

The point of doing this is that $\tilde{G}$ can effectively serve as the public key of the resulting encryption scheme. Anyone who now wants to encrypt a message m of their choice can feed in this message and some randomness to $\tilde{G}$ to generate a ciphertext encrypting m under the key K. Further, the hope of the security was that a secret-key encryption scheme remains secure if one is given an access to an oracle computing encryptions of any message m. And since $\tilde{G}$ is an obfuscation of the encryption function, its security should imply the security of this scheme. This natural approach was formalized and realized using the notion of obfuscation that satisfies the property of providing "as much unintelligiblity as possible" in [SW14].

A public-key encryption scheme is an extremely natural application, but today we have many candidates for public-key encryption schemes whose security rest upon a number of well-studied assumptions. Therefore, realizing an object as basic as a public-key encryption scheme using an obfuscation scheme is really like using a sledge-hammer to open a nut. Let us therefore ask if this primitive might be useful for more advanced applications (without getting to formal details). We discuss a few of them below:

- **Software protection.** The most intuitive application of obfuscation is software protection. Intuitively, since an obfuscation scheme provides unintelligibility, it makes it provably hard for an attacker to learn sensitive information from obfuscated software, while still allowing an honest client to use the software on any inputs of their choice.

  Consider the following example. Let us suppose that Alice is a software developer and she has constructed a really powerful software that she wants to monetize. For doing that she needs to construct a demo version or evaluation version. Doing this "from scratch" is not a good idea: naively, one will have to go over the code line by line and remove fragments that are not supposed to be used in the demo version. This is not an easy task because one can never be sure that in this process of removal no sensitive information is leaked out. This task also involves effort and is sensitive to the judgement of the demo creator. Using obfuscation we can propose an extremely systematic solution: One can take the software and simply "deactivate" certain branches by forming an external wrapper, and then obfuscate the resulting program. This notion is called *crippleware*. Since the sensitive branches are commented out in the obfuscated code, it is as if those branches never existed due to the security of the obfuscation scheme.

- **Obfuscating Machine Learning.** A natural application of obfuscation in the machine learning world is to obfuscate classifiers. Typically, classifiers are trained using large datasets, and revealing the parameters of a classifier can give rise to data privacy issues. Obfuscation is a natural tool to immunize against many such attacks. Formalizing the appropriate security guarantees for obfuscation schemes useful for this purpose as well as properties of the classifiers that are compatible with those obfuscation schemes is an interesting open problem.

- **Cloud Computing**. Another modern cryptographic scenario is that of cloud computing. Typically, when hosting services on a cloud, the service developer creates software which has proprietary information and also creates credentials used to authenticate clients. This information is handed *in the clear* over to the cloud provider which runs the service for the developer. However, if the cloud provider is itself adversarial (for example, due to a hack) then this could seriously jeopardize the developer. Using obfuscation, one can hope for a solution of this problem. In fact, using *indistinguishability obfuscation*, which is the notion we study in this thesis, and is known to provide the best possible unintelligibility guarantees, a solution to this problem was formalized in the work of [BGMS15].

Indistinguishability Obfuscation [BGI+01] (henceforth denoted by $i\mathcal{O}$), is indeed a powerful object. In fact, in the world of cryptography, today we know that $i\mathcal{O}$ can be used to realize almost all known interesting cryptorgaphic primitives including most of the applications discussed above. We will now define what $i\mathcal{O}$ is and talk about how the definition evolved, which is an interesting journey in itself.

1.1 Definition and History

Let us now define what $i\mathcal{O}$ requires. Let $C_0, C_1 : \{0,1\}^n \to \{0,1\}$ be any two Boolean circuits that satisfy the following properties:

- C_0 and C_1 have the same size,
- On every input $x \in \{0,1\}^n$, $C_0(x) = C_1(x)$,

Other than satisfying these two properties, however, C_0 and C_1 may have arbitrary differences in implementation. Then $i\mathcal{O}$ requires that $\tilde{C}_0 = i\mathcal{O}(C_0)$ is computationally indistinguishable to $\tilde{C}_1 = i\mathcal{O}(C_1)$.

This requirement guarantees that $i\mathcal{O}$ hides implementation differences. As a concrete example, let C_0 be (an appropriately padded version of) the bubble sorting program and C_1 be (an appropriately padded version of) the selection sorting program. Then, $i\mathcal{O}$ will guarantee that given an obfuscation of either circuit, it is hard for an adversary to guess if it was actually an obfuscation of bubble sort or selection sort or some other sorting program. More formally, here is the definition:

Definition 1.1.1 (Indistinguishability Obfuscator (iO) for Circuits)**.** *A uniform PPT algorithm $i\mathcal{O}$ is called a secure indistinguishability obfuscator for polynomial-sized circuits if the following holds:*

- ***Completeness**: For every $\lambda \in \mathbb{N}$, every circuit C with input length n, every input $x \in \{0,1\}^n$, we have that*

$$\Pr\left[\tilde{C}(x) = C(x) \;:\; \tilde{C} \leftarrow i\mathcal{O}(1^\lambda, C)\right] = 1 \ .$$

- ***Indistinguishability:** For every two ensembles $\{C_{0,\lambda}\}$ and $\{C_{1,\lambda}\}$ of polynomial-sized circuits that have the same size, input length, and output length, and are functionally equivalent, that is, $\forall\lambda$, $C_{0,\lambda}(x) = C_{1,\lambda}(x)$ for every input x, the following distributions are computationally indistinguishable.*

$$\{i\mathcal{O}(1^\lambda, C_{0,\lambda})\} \qquad \{i\mathcal{O}(1^\lambda, C_{1,\lambda})\}$$

A reader might wonder, why is this definition the most natural one? Indeed, this question took a long time to be answered.

History of Definition. Despite being posed as an open question in [DH76] in 1976, the first systematic analysis of the definitions was done in the beautiful work of Barak et. al. [BGI+01]. The work mainly looked at what is called virtual-black box obfuscation / ideal obfuscation (VBB for short). A VBB obfuscation scheme captures the most intuitive requirement of being as unintelligible as possible: in a VBB scheme, an obfuscation $\tilde{C}$ of any circuit C should give no more knowledge about the circuit C to an adversary than a black-box implementing the circuit C. A black-box implementing C can only respond to input queries x, to which it responds by answering $C(x)$. Therefore, such an obfuscation scheme converts a program to a *virtual black box.*

The work of [BGI$^+$01] unfortunately proved that a general-purpose VBB obfuscation scheme cannot exist. This was mostly seen as a huge negative result for obfuscation back then. At the same time, the same work also gave us a glimmer of hope. They defined $i\mathcal{O}$ as a suggestion for an alternate definition. At the time, it was neither clear that $i\mathcal{O}$ security is the best theoretically achievable guarantee one could hope for, nor was it clear if $i\mathcal{O}$ is feasible to construct, nor was it clear if $i\mathcal{O}$ had any usefulness for applications. All these questions were left open. It also took quite a while for the community to answer these questions.

The answer to the first question came six years later in 2007. The work of Goldwasser and Rothblum [GR07] rekindled the excitement about $i\mathcal{O}$ by proving that $i\mathcal{O}$ is in fact the strongest guarantee that one can hope to achieve. Formally it was proven that $i\mathcal{O}$ is at least as secure as the *best possible obfuscation* scheme that can ever exist. The insight is the following: Let us say that there is another hypothetical obfuscation algorithm $\mathcal{O}$, which achieves the best possible achievable security for whatever purpose. Due to correctness of this best-possible obfuscation $\mathcal{O}$, we have that C is functionally equivalent to $\mathcal{O}(C)$. Because of that, $i\mathcal{O}(C)$ is actually indistinguishable to $i\mathcal{O}(\mathcal{O}(C))$, up to issues of padding the sizes of the circuits. Thus, $i\mathcal{O}(C)$ is as "secure" as $\mathcal{O}(C)$.

A few years later, the work of [SW14] introduced techniques that helped us understand how powerful $i\mathcal{O}$ is. Over the next several years, there was a lot of work establishing numerous applications of $i\mathcal{O}$. Using specially designed proof techniques, we can show that $i\mathcal{O}$ security is enough to realize many of the applications written before. We will discuss this progress in Section 1.2.1.

Unfortunately however, progress on the feasibility front had not been as successful. Even after years of intense study, the state of feasibility of this question was unclear until recently. All known constructions, prior to the results contained in this thesis, relied on assumptions introduced solely for the purpose of proving $i\mathcal{O}$ constructions secure. Many of those assumptions and constructions were subject to cycles of break and repair. A construction of $i\mathcal{O}$ with a sound basis for security had remained elusive. We discuss the huge body of work on this topic in Section 1.4. The main objective of this thesis is therefore to answer the following question:

Feasibility Question (Diffie-Hellman [DH76], Barak et. al. [BGI$^+$01]):

Can we construct $i\mathcal{O}$ which is as hard to break as well-studied hard problems?

We would like to build an $i\mathcal{O}$ scheme whose security rests upon cryptographic assumptions that have stood the test of the time, have a long history of study, and are widely believed to be true. The main result of this thesis is the construction of an $i\mathcal{O}$ scheme from well studied assumptions thereby answering the above feasibility question. We discuss this in more detail next.

1.2 Our Results

The results of this thesis are based on joint work with Rachel Lin and Amit Sahai [JLS21a, JLS21b]. Our main result is a construction of an $i\mathcal{O}$ scheme whose security rests upon the following three well-studied cryptographic assumptions. We prove:

Theorem 1.2.1. *(Informal [JLS21a, JLS21b])* Assume sub-exponential security of the following assumptions:

- the Learning Parity with Noise (LPN) assumption over general prime fields $\mathbb{F}_p$ with polynomially many LPN samples and error rate $1/k^\delta$, where k is the dimension of the LPN secret, and $\delta > 0$ is any constant;
- the existence of a Boolean Pseudo-Random Generator (PRG) in NC^0 with stretch $n^{1+\tau}$, where n is the length of the PRG seed, and $\tau > 0$ is any constant;
- the Decision Linear (DLIN) assumption on symmetric bilinear groups of prime order.

Then, (subexponentially secure) indistinguishability obfuscation for all polynomial-size circuits exists. Further, assuming only polynomial security of the assumptions above yields polynomially secure functional encryption for all polynomial-size circuits.

All the three assumptions are based on computational problems with a long history of study, rooted in complexity, coding, and number theory. Further, they were introduced for building basic cryptographic primitives (such as public key encryption), and have been used for realizing a variety of cryptographic goals that have nothing to do with $i\mathcal{O}$. We discuss all the assumptions in detail in Section 1.3.

This result is a culmination of a long line of work [AJS18, AJL$^+$19, JLMS19, BHJ$^+$19, GJLS21] that led to the result of [JLS21a], which gave a construction of $i\mathcal{O}$ from the above assumptions in addition to the learning with errors assumption [Reg05] which is a well-studied lattice assumption. This result relied on a number of building blocks which were developed during the line of work to build what is known as a functional encryption scheme (which is a primitive known to be simpler than $i\mathcal{O}$ itself: see Definition 2.3.1 for details). The construction of $i\mathcal{O}$ follows from the result of [BV15, AJ15] who showed that a functional encryption scheme can be turned into an $i\mathcal{O}$ scheme under subexponential security loss.

In a followup work [JLS21b] to [JLS21a], the construction was improved in two aspects. First, it was shown that the learning with errors assumption is no longer needed for a secure construction of $i\mathcal{O}$, and second, that functional encryption can be constructed almost immediately relying on just two simple building blocks. Thus, this yields a much simpler and more direct construction. In this thesis we present this simplified view point. We discuss the exact technical framework in Chapter 2.

Another important aspect of our result is that the construction implies feasibility of many other applications, which were previously not known to exist, assuming well-studied hard problems. We discuss this next.

1.2.1 Applications

The notion of $i\mathcal{O}$ occupies an intriguing and influential position in complexity theory and cryptography. Interestingly, if $\mathsf{NP} \subseteq \mathsf{BPP}$, then $i\mathcal{O}$ exists for the class of all polynomial-size circuits, because if $\mathsf{NP} \subseteq \mathsf{BPP}$, then it is possible to efficiently compute a canonical form for any function computable by a polynomial-size circuit. On the other hand, if $\mathsf{NP} \not\subseteq \mathsf{io\text{-}BPP}$, then in fact the existence of $i\mathcal{O}$ for polynomial-size circuits implies that one-way functions exist [KMN+14]. A large body of work has shown that $i\mathcal{O}$ plus one-way functions imply a vast array of cryptographic objects, so much so that iO has been conjectured to be a "central hub" [SW14, KMN+14] for cryptography.

An impressive list of fascinating new cryptographic objects are only known under $i\mathcal{O}$ or related objects such as functional encryption and witness encryption. Hence, our construction of $i\mathcal{O}$ from well-founded assumptions immediately implies these objects from the same assumptions. Below, we highlight a subset of these implications as corollaries. In all the applications, by λ we denote the security parameter.

Corollary 1.2.1 (Informal)**.** *Assume the* subexponential *hardness of the three assumptions in Theorem 1.2.1, we have:*

- *Multiparty non-interactive key exchange in the plain model (without trusted setup), e.g., [BZ14, KRS15].*
- *Adaptively secure succinct garbled RAM, where the size of the garbled program is* $\mathsf{poly}(\lambda, \log T)|P|$ *depending linearly on the description size of the RAM program* P*, the size of the garbled input is* $\mathsf{poly}(\lambda)|x|$ *depending linearly on the size of the input* x*, and evaluation time is quasilinear in the running time of* P *on* x *[BGL+15, CHJV15, KLW15, CCC+15, CH16, CCHR16, ACC+16, AL18].*
- *Indistinguishability obfuscation for RAM, where the size of obfuscated program is* $\mathsf{poly}(\lambda, n, |P|)$ *where* $|P|$ *is the description size of the RAM program* P *and* n *is its input length [BGL+15,*

CHJV15, KLW15, CCC$^+$15, CH16, CCHR16, ACC$^+$16, AL18].

- *Selectively sound and perfectly zero-knowledge Succinct Non-interactive ARGument (SNARG) for any NP language with statements up to a bounded polynomial size in the CRS model, where the CRS size is $\mathsf{poly}(\lambda)(n+m)$, n, m are upper bounds on the lengths of the statements and witnesses, and the proof size is $\mathsf{poly}(\lambda)$ [SW14]*[1].

- *Sender deniable encryption [SW14], and fully deniable interactive encryption [CPP20].*

- *Constant round concurrent zero-knowledge protocols for any NP language [CLP15].*

- *(Symmetric or asymmetric) multilinear maps with boudned polynomial multilinear degrees, following [AFH$^+$16, FHHL18, AMP19], and self-bilinear map over composite and unknown order group, assuming additionally the polynomial hardness of factoring [YYHK14].*

- *Correlation intractable functions for all sparse relations verifiable in bounded polynoimal size, assuming additionally the polynomial hardness of input hiding obfuscators for evasive circuits [CCR16], or for all sparse relations, assuming additionally the exponential optimal hardness of input hiding obfuscators for multibit point functions [KRR17].*

- *Witness Encryption (WE) for any NP language, following as a special case of $i\mathcal{O}$ for polynomial size circuits.*

- *Secret sharing for any monotone function in NP [KNY14].*

- *Fully homomorphic encryption scheme for unbounded-depth polynomial size circuits (without relying on circular security), assuming slighly superpolynomial hardness of*

[1]This construction does not contradict the lower bound result by [GW11] showing that it is impossible to base the adaptive soundness of SNARGs on falsifiable assumptions via black-box reductions, since this construction only achieves selective soundness.

the assumptions above [CLTV15].

Corollary 1.2.2 (Informal). *Assume the* polynomial *hardness of the three assumptions in Theorem 1.2.1, we have:*

- *Attribute Based Encryption (ABE) for unbounded-depth polynomial-size circuits, following as a special case of functional encryption for unbounded-depth polynomial size circuits.*
- *Hard instances for finding Nash Equilibrium (more generally for the class* PPAD*) [AKV04, BPR15, GPS16, HY17, KS17].*

Regarding PPAD hardness, another line of beautiful works [CHK$^+$19a, CHK$^+$19b, CCH$^+$19, EFKP20, LV20, JKKZ20] showed that the hardness of #SAT reduces to that of PPAD, assuming the adaptive soundness of applying Fiat-Shamir to certain protocols. Most recently, this led to basing the PPAD hardness on that of #SAT and the sub-exponential LWE assumption [JKKZ20]. In comparison, relying on [GPS16], using our Functional Encryption construction, we can base the PPAD hardness on the polynomial security of the three assumptions we make. Next, we discuss another important aspect of our work. Note that none of the three assumptions in Theorem 1.2.1 are lattice assumptions. Since $i\mathcal{O}$ has wide applicability, this also gives first constructions of some of these primitives that were previously known only via lattice based hardness assumptions.

1.2.1.1 Homomorphic Encryption without Lattices

The advent of lattice-based techniques have led to a revolution in cryptography. The most notable herald of the lattice era came with the pioneering work of Gentry [Gen09] constructing Fully Homomorphic Encryption (FHE) using ideal lattices, followed by the breakthrough result of Brakerski and Vaikuntanathan [BV11] constructing FHE from the Learning With Errors (LWE) assumption. To this day, the only known constructions of FHE are based on the hardness of lattice-type problems – either directly from problems like LWE or Ring LWE, or slightly indirectly via problems such as the approximate GCD [vDGHV10]. Beyond FHE, lattice techniques have also been used in all known proposals for candidate multilinear maps [GGH13a, CLT13, GGH15]. All surviving proposals for achieving indistinguishability obfuscation ($i\mathcal{O}$) that we are aware of also use lattice-based problems (see [JLS21a] and the references therein). The common thread to all problems based on lattices is that they utilize *small* noise where the *smallness* is measured using the standard notion of absolute value over $\mathbb{R}$. Leveraging the smallness of noise, lattice problems and techniques have been at the heart of nearly every work over the past decade attempting to achieve advanced cryptographic feasibility goals.

We observe that none of these assumptions in Theorem 1.2.1 involve any notion of smallness related to absolute value over $\mathbb{R}$, and in fact no lattice-based methods are known to be useful in either manipulating or attacking these assumptions. Furthermore, of the three assumptions in Theorem 1.2.1, only one of them is known to imply public-key encryption or key agreement on its own – the DLIN assumption. Even assuming the other two assumptions simultaneously, it is not known how to build key agreement or any other "public key" primitive. (Recall that known constructions of public-key encryption from LPN require $\delta \geq \frac{1}{2}$ in our language above [Ale03, AAB15].) Thus, to the best of our knowledge, the other two assumptions appear to be qualitatively weaker than – in the sense of less powerful than – DLIN, and indeed weaker than LWE or other such commonly used lattice-based assumptions.

An immediate consequence of our theorem is that the combination of bilinear pairing, LPN over $\mathbb{F}_p$, and constant-locality PRG is sufficient for building all the primitives that are implied by $i\mathcal{O}$ or Functional Encryption (FE) (and other assumptions that are implied by one of the three assumptions). This, somewhat surprisingly, includes *Fully Homomorphic Encryption* (FHE) that support homomorphic evaluation of (unbounded) polynomial-size circuits, through the construction by [CLTV15] that shows FHE can be built from subexponentially secure $i\mathcal{O}$ and rerandomizable encryption, which is implied by the DLIN assumption. It also includes Attribute Based Encryption (ABE) that support policies represented by (unbounded) polynomial-size circuits, which is a special case of functional encryption. To this day, the only known constructions of FHE and ABE for circuits are based on the hardness of lattice-type problems – either directly from problems like LWE or Ring LWE, or slightly indirectly via problems such as the approximate GCD problem [vDGHV10]. Our work hence yields the first alternative pathways towards these remarkable primitives. Thus,

Corollary 1 (Informal)**.** *Assume the same assumptions as in the Theorem 1.2.1. Then, fully homomorphic encryption for all polynomial-sized circuits exist.*

We emphasize that our result complements instead of replaces lattice-based constructions. It also gives rise to several exciting open directions for future work, such as, can we obtain *direct* constructions of FHE or ABE (not via $i\mathcal{O}$ or FE) from the trio of assumptions? and is there any formal relationship between these assumptions and lattice assumptions (e.g, BDD, SVP etc.)?

1.3 Assumptions in More Detail

We now describe each of the assumptions we need in more detail and briefly survey their history.

The DLIN Assumption: The Decisional Linear assumption (DLIN) is stated as follows: For an appropriate λ-bit prime p, two groups $\mathbb{G}$ and $\mathbb{G}_T$ are chosen of order p such that there exists an efficiently computable nontrivial symmetric bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. A canonical generator g for $\mathbb{G}$ is also computed. Then, the DLIN assumption requires that the following computational indistinguishability holds:

$$\{(g^x, g^y, g^{xr}, g^{ys}, g^{r+s}) \mid x, y, r, s \leftarrow \mathbb{Z}_p\}$$
$$\approx_c \{(g^x, g^y, g^{xr}, g^{ys}, g^z) \mid x, y, r, s, z \leftarrow \mathbb{Z}_p\}$$

This assumption was first introduced in the 2004 work of Boneh, Boyen, and Shacham [BBS04]. Since then DLIN and assumptions implied by DLIN have seen extensive use in a wide variety of applications throughout cryptography, such as Identity-Based Encryption, Attribute-Based Encryption, Functional Encryption for degree two polynomials, Non-Interactive Zero Knowledge, etc. (See, e.g. [GS08, BKKV10, OT10, BJK15, JR13]).

The existence of PRGs in NC^0: The assumption of the existence of a Boolean Pseudo-Random Generator PRG in NC^0 states that there exists a Boolean function $\mathsf{G} : \{0,1\}^n \to \{0,1\}^m$ where $m = n^{1+\tau}$ for some constant $\tau > 0$, and where each output bit computed by G depends on a constant number of input bits, such that the following computational indistinguishability holds:

$$\{\mathsf{G}(\sigma) \mid \sigma \leftarrow \{0,1\}^n\} \approx_c \{\mathbf{y} \mid \mathbf{y} \leftarrow \{0,1\}^m\}$$

Pseudorandom generators are a fundamental primitive in their own right, and have vast applications throughout cryptography. PRGs in NC^0 are tightly connected to the fundamental topic of Constraint Satisfaction Problems (CSPs) in complexity theory, and were first proposed for cryptographic use by Goldreich [Gol00, CM01, IKOS08] 20 years ago. The complexity theory and cryptography communities have jointly developed a rich body of literature on the cryptanalysis and theory of constant-locality Boolean PRGs [Gol00, CM01, MST03, CEMT09, BQ09, ABW10, ABR12, BQ12, App12, App13, OW14, AL16, KMOW17, CDM+18, AK19].

LPN over prime fields: The Learning Parity with Noise LPN assumption over general prime fields $\mathbb{F}_p$ is a decoding problem. The standard LPN assumption with respect to subexponential-size modulus p, dimension ℓ, sample complexity n, and a noise rate $r = 1/\ell^{\delta}$ for some $\delta \in (0, 1)$, states that the following computational indistinguishability holds:

$$\{\mathbf{A}, \mathbf{s} \cdot \mathbf{A} + \mathbf{e} \bmod p \mid \mathbf{A} \leftarrow \mathbb{Z}_p^{\ell \times n},\ \mathbf{s} \leftarrow \mathbb{Z}_p^{1 \times \ell},\ \mathbf{e} \leftarrow \mathcal{D}_r^{1 \times n}\}$$
$$\approx_c \{\mathbf{A}, \mathbf{u} \mid \mathbf{A} \leftarrow \mathbb{Z}_p^{\ell \times n},\ \mathbf{u} \leftarrow \mathbb{Z}_p^{1 \times n}\}.$$

Above $e \leftarrow \mathcal{D}_r$ is a generalized Bernoulli distribution, *i.e.* e is sampled randomly from $\mathbb{Z}_p$ with probability $1/\ell^{\delta}$ and set to be 0 with probability $1 - 1/\ell^{\delta}$. We consider polynomial sample complexity $n(\ell)$, and the modulus p is an arbitrary subexponential function in ℓ.

The origins of the LPN assumption date all the way back to the 1950s: the works of Gilbert [Gil52] and Varshamov [Var57] showed that random linear codes possessed remarkably strong minimum distance properties. However, since then, almost no progress has been made in efficiently decoding random linear codes under random errors. The LPN over fields assumption above formalizes this, and was introduced over $\mathbb{Z}_2$ for cryptographic uses in 1994 [BFKL94], and formally defined for general finite fields and parameters in 2009 [IPS08], under the name "Assumption 2".

While in [IPS08], the assumption was used when the error rate was assumed to be a constant, in fact, polynomially low error (in fact $\delta = 1/2$) has an even longer history in the LPN literature: it was used by Alekhnovitch in 2003 [Ale03] to construct public-key encryption with the field $\mathbb{Z}_2$, and used to build public-key encryption over $\mathbb{Z}_p$ in 2015 [AAB15]. The exact parameter settings that we describe above, with both general fields and inverse polynomial error rate corresponding to an arbitrarily small constant $\delta > 0$ was explicitly posed by [BCGI18], in the context of building efficient secure two-party and multi-party protocols for arithmetic computations.

Recently, the LPN assumption has led to a wide variety of applications (see for example, [IPS08, AAB15, BCGI18, ADI+17, DGN+17, GNN17, BLMZ19, BCG+19]). A comprehensive review of known attacks on LPN over large fields, for the parameter settings we are interested in, was given in [BCGI18, BCG+20]. For our parameter setting, the running time of all known attacks is $\Omega(2^{\ell^{1-\delta}})$, for any choice of the constant $\delta \in (0,1)$ and for any polynomial number of samples $n(\ell)$.

On search vs. decision versions of our assumptions. Except for the DLIN assumption, the other three assumptions that we make can be based on search assumptions.

The LPN over $\mathbb{Z}_p$ assumption we require are implied by the subexponential hardness of their corresponding search versions [MM11, BFKL94, MP13, Reg05]. As summarized in [Vai20], there is a search-to-decision reduction[2] whose sample complexity is $m = \mathsf{poly}(\dim(\mathbf{s}), m', 1/\epsilon)$ (namely, polynomial in the dimension $\dim(\mathbf{s})$ of the secret, sample complexity m' of the decision version, and the inverse of the distinguishing gap ϵ), and runtime $\mathsf{poly}(\dim(\mathbf{s}), p, m)$. In this work, we need the pseudorandomness of (polynomially many) LPN samples to hold against polynomial-time adversaries, with a *subexponential* distinguishing gap. We can further set the modulus p to an arbitrarily small subexponential function in $\dim(\mathbf{s})$[3]. Decisional LPN with such parameters are implied by the subexponential search LPN assumptions: There is a constant $\gamma > 0$ such that no subexponential-time $2^{\dim(\mathbf{s})^\gamma}$ adversary, given a subexponential $2^{\dim(\mathbf{s})^\gamma}$ number of samples, can recover $\mathbf{s}$ with noticeable probability.

The works of [App13, AK19] showed that the one-wayness of *random local functions* implies the existence of PRGs in NC^0. More precisely, for a length parameter $m = m(n)$, a locality parameter $d = O(1)$, and a d-ary predicate $Q : \{0,1\}^d \to \{0,1\}$, a distribution $\mathcal{F}_{Q,m}$ samples a d-local function $f_{G,Q} : \{0,1\}^d \to \{0,1\}$ by choosing a random d-uniform hypergraph G with n nodes and m hyperedges, where each hyperedge is chosen uniformly and independently at random. The i^{th} output bit of $f_{G,Q}$ is computed by evaluating Q on the d input bits indexed by nodes in the i^{th} hyperedge. The one-wayness of $\mathcal{F}_{Q,m}$ for proper choices of Q, m has been conjectured and studied in [Gol00, MST03, CEMT09, BQ09, ABW10]. The

[2]Importantly, this reduction is oblivious to the distribution of the errors and hence applies to both LWE and LPN.

[3]In the construction, we set $p = \Theta(2^\lambda)$ and $\dim(\mathbf{s})$ to a large enough polynomial in λ.

works of [App13, AK19] showed how to construct a family of PRG in NC^0 with polynomial stretch based on the one-wayness of $\mathcal{F}_{Q,m}$ for any Q that is sensitive (i.e., some input bit i of Q has full influence) and any $m = n^{1+\delta}$ with $\delta > 0$. The constructed PRGs have negligible distinguishing advantage and the reduction incurs a multiplicative polynomial security loss. Therefore, the subexponential pseudorandomness of PRG in NC^0 that we need is implied by the existence of $\mathcal{F}_{Q,m}$ that is hard to invert with noticable probability by adversaries of some subexponential size.

1.4 Prior Work on Feasibility of $i\mathcal{O}$

There is a rich landscape of research on conjectured constructions of $i\mathcal{O}$. Despite being posed as a question at least 20 years ago [DH76, BGI$^+$01], the first candidate mathematical construction came only in 2013, through the work of [GGH$^+$13b]. This construction relied on a newly constructed primitive called multilinear maps [GGH13a], which is a generalization of a bilinear maps where one could compute high degree computations in the exponents. Soon after, several different candidates for multilinear maps were proposed [CLT13, GGH15] and many other constructions of $i\mathcal{O}$ were proposed. This propelled a huge body of constructions of $i\mathcal{O}$ relying on multilinear maps and related ideas (e.g. [GGH13a, GGH$^+$13b, BGK$^+$14, BR14, PST14, BMSZ16, CLT13, GGH15, CLR15, MF15, MSZ16, DGG$^+$16].) Unfortunately, all these works suffered from one of the three main problems:

- Most constructions were heuristic in the sense that they were just conjectured to be secure. There was no simple assumption on the multilinear maps on which you can base security,

- Sometimes, security was based on some new assumption, but it was a new assumption proposed solely for the proving the construction secure. Such assumptions lacked a long history of study,

- Most of the time, in both the above cases there were actually cycles of attacks and fixes on the constructions and/or underlying assumptions (e.g. [CHL+15, HJ15, CLR15, MF15, MSZ16, BBKK18, LV17, BHJ+19]) which reduced our confidence further.

With this the focus shifted to trying to minimize the degree of the multilinear map needed, with the goal of eventually reaching degree 2. In a beautiful line of work [Lin16, LV16, AS17, Lin17, LT17] it was shown that $i\mathcal{O}$ can be constructed just from succinct assumptions on degree-3 multilinear maps. Unfortunately, the candidates for degree-3 multilinear maps were the same as the candidates for high degree multilinear maps and suffered from the same class of attacks as before.

Soon after, a line of work [AJS18, Agr19, LM18, AJL+19, JLMS19, AP20, GJLS21] constructed $i\mathcal{O}$ relying on bilinear maps, along with new kinds of pseudorandom generators. These assumptions were much simpler to state than before. Even though earlier proposals for some of those pseudorandom generators were attacked [LV17, BBKK18, BHJ+19], exploring the limits of those attacks helped us design $i\mathcal{O}$ based on new but simple-to-state assumptions [AJL+19, JLMS19, GJLS21] that resisted all known attacks. Unfortunately however, these assumptions were newly stated and did not have a long history of study.

Therefore, building upon [AJS18, LM18, AJL+19, JLMS19, GJLS21] these works culminated finally in our recent works [JLS21a, JLS21b], which managed to construct $i\mathcal{O}$ from the three assumptions in Theorem 1.2.1. This eliminated the need for making any new unstudied hardness assumptions. We now discuss some of the main open problems in the space of $i\mathcal{O}$ constructions.

1.5 Open Problems

Our work places $i\mathcal{O}$ on firm foundations with respect to the assumptions it is based on, thereby answering the main feasibility question for the primitive (until we resolve the P vs. NP question). However, there are many important open questions that remain to be answered.

- **Concrete Efficiency:** Our work first builds the notion of functional encryption and then boosts this object to $i\mathcal{O}$ via a complex transformation [AJ15, BV15]. As a result, the final construction is quite complex. A highly important question that remains open is: Is it possible to construct $i\mathcal{O}$ either by fine tuning our approach, or otherwise (as in [GJK18, BIJ$^+$20]) in a way that the resulting scheme yields concrete implementable efficiency? For this question, as a first step, it is even interesting if the construction rests upon new assumptions as long as the assumptions are rigorously cryptanalyzed.

- **Post-Quantum $i\mathcal{O}$:** Our work relies on Bilinear maps (in a somewhat crucial way). As a result of that, our construction is broken in polynomial time using a quantum computer. Therefore, an important and a natural question to ask here is if we can build $i\mathcal{O}$ on any combination of well-studied post-quantum assumptions such as $\mathsf{LWE}, \mathsf{LPN}$, or PRG in NC^0. This is indeed an active area of research.

- **$i\mathcal{O}$ for Quantum Circuits:** All known constructions of $i\mathcal{O}$ support only classical circuits. If quantum computers come one day, an interesting question is to construct an $i\mathcal{O}$ scheme that can be used to actually obfuscate quantum circuits. There are some results in restricted models [BK20, BM21] but none of the known constructions work to obfuscate general quantum programs.

- **Understanding Assumptions Better:** We are still in the early stages of understanding the feasibility of $i\mathcal{O}$. An immediate question that arises out of work is to identify essential and non-essential assumptions out of the three assumptions, and, if any of the assumptions can be replaced by another one. Identifying if there is any other substantially different approach that also yields $i\mathcal{O}$ from well-studied assumptions will also shed light on this question.

1.6 Organization

As described before, the result in [JLS21a, JLS21b] is a culmination of a line of work [AJS18, LM18, Agr19, AJL$^+$19, JLMS19, BHJ$^+$19, GJLS21]. These results identified several building blocks required to build $i\mathcal{O}$ and constructed each of them from various assumptions. Many of these building blocks employed concepts from a number of areas in cryptography and theoretical computer science such as pseudorandomness, lattice based cryptography, elliptic curve cryptography, privacy amplification, coding theory and even convex optimization algorithms. In this thesis, we will present complete technical details of the construction largely based upon [JLS21b] which is considerably more direct and simpler in comparison with earlier works and relies on just two simple building blocks. We build one of the building block from LPN assumption and PRG in NC^0 and the other buidling block from the DLIN assumption. In Chapter 2 we give an overview of the technical framework and define these two abstractions. Later, in the respective chapters we construct these notions.

CHAPTER 2

Technical Roadmap

In this chapter, we will describe our high-level approach to build $i\mathcal{O}$. This approach will require us to consider several intermediate blocks. We will also motivate and formally define each of those notions. We will also discuss a bit of history behind these objects, and finally in the respective future chapters we will see their constructions. Before we proceed further we set up some preliminaries to be used in the rest of the chapters.

2.1 Preliminaries

We now set up some notations that will be used throughout the thesis. Throughout, we will denote the security parameter by λ. For any distribution $\mathcal{X}$, we denote by $x \leftarrow \mathcal{X}$ the process of sampling a value x from the distribution $\mathcal{X}$. Similarly, for a set X we denote by $x \leftarrow X$ the process of sampling x from the uniform distribution over X. For an integer $n \in \mathbb{N}$ we denote by $[n]$ the set $\{1, .., n\}$. Throughout, when we refer to polynomials in security parameter, we mean constant degree polynomials that take positive value on non-negative inputs. We denote by $\mathsf{poly}(\lambda)$ an arbitrary polynomial in λ satisfying the above requirements of non-negativity.

We use standard Landau notations. We will also use $\widetilde{O}$, where for any function $a(n, \lambda)$, $b(n, \lambda)$, we say that $a = \widetilde{O}(b)$ if $a(n, \lambda) = O(b(n, \lambda)\, \mathsf{poly}(\lambda, \log_2 n))$ for some polynomial poly. A function $\mathsf{negl} : \mathbb{N} \to \mathbb{R}$ is negligible if $\mathsf{negl}(\lambda) = \lambda^{-\omega(1)}$. Further, the negl is subexponentially small if $\mathsf{negl}(\lambda) = 2^{-\lambda^{\Omega(1)}}$.

We denote vectors by bold-faced letters such as $\mathbf{b}$ and $\mathbf{u}$. Matrices will be denoted by capitalized bold-faced letters for such as $\mathbf{A}$ and $\mathbf{M}$. For any $k \in \mathbb{N}$, we denote by the notation $\mathbf{v}^{\otimes k} = \underbrace{\mathbf{v} \otimes \cdots \otimes \mathbf{v}}_{k}$ the standard tensor product. This contains all the monomials in the variables inside $\mathbf{v}$ of degree exactly k.

Multilinear Representation of Polynomials and Representation over $\mathbb{Z}_p$. A straightforward fact from analysis of boolean functions is that every NC^0 function $F : \{0,1\}^n \to \{0,1\}$ can be represented by a unique constant degree multilinear polynomial $f \in \mathbb{Z}[\mathbf{x} = (x_1, \ldots, x_n)]$, mapping $\{0,1\}^n$ to $\{0,1\}$. At times, we consider a mapping of such polynomial $f \in \mathbb{Z}[\mathbf{x}]$ into a polynomial g over $\mathbb{Z}_p[\mathbf{x}]$ for some prime p. This is simply obtained by reducing the coefficients of f modulo p and then evaluating the polynomial over $\mathbb{Z}_p$. Observe that $g(\mathbf{x}) = f(\mathbf{x}) \mod p$ for every $\mathbf{x} \in \{0,1\}^n$ as $f(\mathbf{x}) \in \{0,1\}$ for every such $\mathbf{x}$. Furthermore, given any NC^0 function F, finding these representations take polynomial time.

Computational Indistinguishability. We now describe how computational indistinguishability is formalized.

Definition 2.1.1 (ϵ-indistinguishability)**.** *We say that two ensembles $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda \in \mathbb{N}}$ and $\mathcal{Y} = \{\mathcal{Y}_\lambda\}_{\lambda \in \mathbb{N}}$ are ϵ-indistinguishable where $\epsilon : \mathbb{N} \to [0,1]$ if for every probabilistic polynomial time adversary $\mathcal{A}$ it holds that: For every sufficiently large $\lambda \in \mathbb{N}$,*

$$\left| \Pr_{x \leftarrow \mathcal{X}_\lambda}[\mathcal{A}(1^\lambda, x) = 1] - \Pr_{y \leftarrow \mathcal{Y}_\lambda}[\mathcal{A}(1^\lambda, y) = 1] \right| \leq \epsilon(\lambda).$$

We say that two ensembles are computationally indistinguishable if they are ϵ-indistinguishable for $\epsilon(\lambda) = \mathsf{negl}(\lambda)$ for some negligible negl, and that two ensembles are sub-exponentially indistinguishable if they are ϵ-indistinguishable for $\epsilon(\lambda) = 2^{-\lambda^c}$ for some positive real number c.

2.2 High-Level Approach

There have been many candidate constructions of $i\mathcal{O}$ from a number of different assumptions. Typically these constructions fall into two categories: i) *Direct Constructions* where a circuit is directly obfuscated relying on suitable cryptographic gadgets such as multilinear maps, and, ii) *Bootstrapping from simpler primitives*, where a seeming simpler object is constructed which is then bootstrapped to $i\mathcal{O}$. Although the approach of constructing obfuscating programs directly may seem more intuitive, arguing security has proven to be really tough. As of now all known direct constructions only give heuristic security.

The second approach, although seemingly more indirect, has been more successful in constructing $i\mathcal{O}$ from simple to state assumptions, since the goal is to construct something simpler. All the works culminating into thesis [AJL$^+$19, JLMS19, GJLS21, JLS21a, JLS21b] follow the second approach.

Perhaps a really important point in our search for Indistinguishability Obfuscation was the beautiful insight produced in [AJ15, BV15]. Both the works concurrently showed that in order to build Indistinguishability Obfuscation it suffices to build something seemingly much simpler - "Sublinear Functional Encryption" (denoted by FE for short). Our contribution is in fact, to construct such an FE using only well-studied hard problems.

Sublinear Functional Encryption. Let us now focus on the key object that suffices to build $i\mathcal{O}$. Intuitively speaking, a functional encryption scheme is a generalization of an encryption scheme. In a traditional encryption scheme, given a ciphertext $\mathsf{CT}(x)$ encrypting a plaintext x, the encryption provides all or nothing access to x. This is because if one has the secret key SK, one can learn x in its entirety, otherwise nothing at all. In a functional encryption scheme, one can be given fine grained access to the data. One can be issued "functional keys" SK_f that are associated with functions $f \in \mathcal{F}$ in some function class. Then, given $\mathsf{CT}(x)$ and SK_f one can learn $f(x)$, but only that. More formally, the scheme ensures the following basic property: let x_0, x_1 be two plain-texts such that $f(x_0) = f(x_1)$, then even given SK_f, $\mathsf{CT}(x_0)$ is computationally indistinguishable to $\mathsf{CT}(x_1)$. This object is called functional encryption. If such a functional encryption scheme satisfies a few additional properties, then it can be used to realize $i\mathcal{O}$. To construct $i\mathcal{O}$, the functional encryption scheme needs to satisfy three additional conditions below:

- The scheme should allow issuing just one functional key from a sufficiently expressive function class, say NC^1 circuits [1],
- (Sublinear Encryption Time): The size of the circuit encrypting a plain-text $\mathbf{x}$ should be bounded by $(s_{\mathcal{F}}^{1-\epsilon} + |\mathbf{x}|)\,\mathsf{poly}(\lambda)$ for some constant $\epsilon > 0$, where s is the maximum size of the circuit in the function class and λ is the security parameter,
- (Subexponential Security): The maximum probability with which a polynomial time adversary can distinguish between an encryption of x_0 from an encryption of x_1 should be bounded by $2^{-\lambda^c}$ for some constant $c > 0$.

In this thesis, we denote by a sublinear functional encryption scheme, a functional encryption scheme satisfying all the properties above. The word "Sublinear" in "Sublinear Functional Encryption" is used to stress the second requirement. The formal definition of this primitive can be found in Section 2.3.

Constructing Sublinear Functional Encryption. In order to build sublinear functional encryption for circuits, our high-level approach is the following: We use a functional encryption scheme FE_{simple} for a simple class of functions $\mathcal{C}_{simple}$ and bootstrap it to a sublinear functional encryption scheme for all circuits. In order to do so, a very natural approach in the past has been to rely on what is called as a sublinear Randomized Encoding Scheme

[1]We consider a related function class, which is also sufficient.

(which we will denote by RE). Recall that in an RE scheme, given an input $\mathbf{x}$ and a randomness $\mathbf{r}$ say in $\{0,1\}^n$ and letting $\mathbf{x}' = (\mathbf{x}, \mathbf{r})$, for any circuit C of size say $n^{1+\epsilon}$ for some $\epsilon > 0$, one can compute $\mathsf{Encode}(C, \mathbf{x}') \to \widehat{C(\mathbf{x})}$. This encoding has the following properties:

- The length of $\widehat{C(\mathbf{x})}$ is $\widetilde{O}(|C| + |x|)$,
- Given $\widehat{C(\mathbf{x})}$, one can decode $C(\mathbf{x})$ in polynomial time,
- $\widehat{C(\mathbf{x})}$ is simulatable knowng just $C, C(\mathbf{x})$, and,
- $\mathsf{Encode}(C, \cdot)$ is usually a low depth circuit. In particular, assuming $\mathsf{PRG} \in \mathsf{NC}^0$ it can even be in NC^0 (for example, [AIK04]).

The word "sublinear" is used to emphasize that the length of the randomness $\mathbf{r}$ is sublinear in the size of the circuit.

This gives us a natural approach. If we have a functional encryption scheme FE_{simple} for the circuit class containing NC^0 circuits and satisfying certain sublinearity properties, then we can already construct full fledged sublinear functional encryption scheme. The idea is that in order to compute encryption of $\mathbf{x}$, we will simply encrypt $\mathbf{x}' = (\mathbf{x}, \mathbf{r})$ using FE_{simple}, and to generate keys for a circuit C, we will generate keys for NC^0 functions $\mathsf{Encode}(C, \cdot)$. To ensure that the resulting FE scheme satisfies sublinear encryption time, we need that the running time to compute FE_{simple} encryption of $\mathbf{x}'$ is sublinear in the size of the circuit C. NC^0 circuits are extremely simple functions. When written as a polynomial over $\mathbb{Z}$, every output can be computed by a polynomial of constant degree (say d).

Using Degree-2 FE. Before our works, the best known functional encryption schemes supported evaluation of degree two polynomials over the encrypted input. Such schemes (for example, the scheme in [Lin17]) can be constructed assuming well-studied assumption on bilinear maps. These schemes support an unbounded number of function keys and are also known to satisfy strong form of encryption efficiency: the encryption time can be linear in

$|\mathbf{x}'|$. One could have solved the problem already if one could construct a sublinear randomized encoding scheme with degree $d = 2$. As one could use a degree two FE to isntantiate our FE_{simple}, and use it along with the degree two randomized encoding.

Unfortunately, such sublinear randomized encodings RE are typically constructed by first constructing a "vanilla" randomized encoding scheme in NC^0, where the length of the randomness can be linear in the size of the circuit, and then composing it with a PRG in NC^0 with polynomial stretch. Such "vanilla" randomized encoding schemes can be constructed in multiple ways (for example [AIK06, Yao86]). The idea now is that the PRG can expand a shorter random string $\mathbf{r} \in \{0,1\}^n$ into a longer pseudorandom string $\mathbf{r}'$, which can then used along with a "vanilla" randomized encoding scheme.

As a result of this, in all current approaches, the complexity of RE is tied innately to constructions of Boolean PRGs with polynomial stretch. Therefore, their degree is at least the degree of the Boolean PRG they rely on. More so, for degree $d = 2$, PRG's with polynomial stretch cannot exist. To see this, observe that any Boolean function that has a multilinear degree d, can depend on at most $d \cdot 2^{d-1}$ variables [NS92]. Thus, for $d = 2$ each output is a function of at most four variables. The work of [MST03] showed that any PRG where each output bit depends on four variables cannot satisfy $m \geq 24n + 1$. Because of this a construction of RE with degree 2 is unlikely to exist. In the past, to escape such implausibility various models have been proposed. One such approach is discussed next.

Allowing Preprocessing. A natural suggestion to bypass this hurdle is to allow preprocessing. Namely, we are asking for an algorithm $\mathsf{PreProc}$ that takes as input $(\mathbf{x}, \mathbf{r})$ and outputs a preprocessing $\widetilde{\mathbf{x}}$, such that:

- Time to compute $\mathsf{PreProc}(\mathbf{x}, \mathbf{r})$ is sublinear in the size of the circuit C,
- Given $\mathsf{Encode}(C, \widetilde{\mathbf{x}}) = \widehat{C(\mathbf{x})}$, one can decode $C(\mathbf{x})$ in polynomial time,
- $\widehat{C(\mathbf{x})}$ is simulatable knowing just $C, C(\mathbf{x})$, and,

- $\mathsf{Encode}(C, \cdot)$ is computable by degree $d = 2$ polynomials over $\widetilde{\mathbf{x}}$.

The reason for suggesting this is that with added relaxation one could possibly preprocess $\mathbf{x}' = (\mathbf{x}, \mathbf{r})$ appropriately, it might be possible to reduce the degree of the computation. We can also readily see that such a notion would be helpful because now one can first preprocess $\mathbf{x}, \mathbf{r}$ into $\widetilde{\mathbf{x}}$ and then encrypt $\mathsf{FE}_{simple}(\widetilde{\mathbf{x}})$ to compute the encryptions. The sublinearity property of the resulting scheme holds because the time to preprocess $\mathbf{x}, \mathbf{r}$ is sublinear in the size of the circuit C, and then in the known quadratic functional encryption scheme, the time to encrypt a message is just linear in the length of the message $\widetilde{\mathbf{x}}$. Therefore, this will result in a sublinear FE scheme.

However, the issue with this approach is that the first requirement proves to be really difficult to match. Indeed, if there is no restriction the time of preprocessing, there is a trivial way to preprocess: precompute all monomials in $\mathbf{x}, \mathbf{r}$ of degree $d/2$. Such a model will not be useful to us.

More formally, this notion has been studied in the special case of a Boolean PRG. The works of [BBKK18, LV17], showed that popular variants of degree-2 preprocessing PRG's (Block-local PRG's [LT17]) are implausible.

Fortunately, [AJS18, AJL$^+$19, JLMS19, GJLS21, JLS21a] suggested a different preprocessing model that bypasses this implausibility. In this work, we will consider this model. We consider a very simple improvement to the function class of degree two computations - handling constant degree public computations in addition to the degree two private computations. We first describe the properties of corresponding FE_{simple}, and then come back to the model of preprcoessing for the randomized encoding scheme. This FE_{simple} is called as a Partially Hiding Functional Encryption scheme, and can be constructed from well-studied assumptions over Bilinear maps [JLMS19, Wee20, GJLS21]. The corresponding randomized encoding which is the one of the main contribution of this work is called as a preprocessed randomized encoding scheme PRE. We construct it from LPN and PRG in NC^0 assumptions.

Partially-Hiding FE. A PHFE scheme is a generalization of a functional encryption scheme generalizing the class of functions slightly beyond quadratic functions while being still constructible using bilinear maps. In particular, now the message could be of the form $(\mathsf{P}, \mathsf{S}) \in \mathbb{Z}_p^n \times \mathbb{Z}_p^n$ where p is the order of bilinear map (think of it as a λ bit prime). The functions that can be handled are of the form:

$$f(\mathsf{P}, \mathsf{S}) = \sum_{j,k} f_{j,k}(\mathsf{P}) \cdot \mathsf{S}_j \cdot \mathsf{S}_k,$$

where each $f_{j,k} : \mathbb{Z}_p^n \to \mathbb{Z}_p$ is a constant degree polynomial over $\mathbb{Z}_p$. The decryption produces $v = f(\mathsf{P}, \mathsf{S}) \mod p$ as long as the value v when interpreted as an integer is bounded by some polynomial in n in absolute value. This might seem as general as constant degree polynomials but the main difference here is that P is just revealed in the clear and is not required be hidden by the encryption. Thus a PHFE scheme allows encrypting plain-texts that has two components: A public input P, and a private input S, and it supports key queries for functions f that are constant degree polynomials over $\mathbb{Z}_p$ with a restriction that their degree in the private input S is bounded by two. This requirement of having degree two in the secret component is what enables a construction from standard assumptions on bilinear maps. To be useful to our goal, the scheme will also satisfy some efficiency requirements:

- It will support an arbitrary polynomial number of key queries,
- The size of the encryption circuit is of size $\widetilde{O}(n)$ where the input length is n,

These two properties are essential properties that enable the resulting functional encryption scheme to satisfy the sublinearity property. We provide the formal definition of this primitive in Section 2.4.

It was shown in [AJS18, LM18, AJL$^+$19, JLMS19, GJLS21, Wee20] that such a scheme can be constructed using well-studied assumptions on bilinear maps (in particular, [Wee20] showed that it can be constructed using DLIN assumption over symmetric bilinear groups). We recall this construction in Chapter 6. For the rest of the thesis and the section below, we call the function class supported by PHFE scheme as degree-$(O(1), 2)$ polynomials over $\mathbb{Z}_p$ indicating its degree in the public and the secret input.

Preprocessed Randomized Encoding Once we have defined a PHFE scheme, the corresponding PRE scheme is straightforward to define.

- The $\mathsf{PreProc}$ algorithm on input $\mathbf{x}, \mathbf{r}$ should output two vectors P, S over $\mathbb{Z}_p$ in time sublinear in the size of the circuit C,
- For any circuit C, $\widehat{C(\mathbf{x})} = \mathsf{Encode}(C, (\mathsf{P}, \mathsf{S}))$ should be computable by degree $(O(1), 2)$ polynomials, and,
- $\mathsf{P}, \widehat{C(\mathbf{x})}$ should be computationally simulatable knowing $C, C(\mathbf{x})$.

Our idea is therefore to observe that this modest looking change to the preprocessing model (allowing for a public input in the preprocessing model) allows us to bypass the implausibility arguments.

Looking ahead, we will take a degree d randomized encoding for any constant $d > 0$ and somehow transfer the complexity to the public input P, so that the resulting computation by degree $(O(d), 2)$ polynomial. For this, we will additionally use LPN assumption.

2.2.1 How to Construct Preprocessed Randomized Encoding

Let us now discuss how to construct a preprocessed randomized encoding scheme, but before we proceed further we want to discuss what such a PRE scheme imply:

Time succinctness vs size succinctness: A reader could ask what if we relax the requirement of the running time to compute the preprocessed input (P, S) to be sublinear in the size of C to requiring only its size be sublinear (where computing time can grow linearly in the size of circuit C). Note that this is actually enough for us, but additionally relying on

LWE: it was shown in [LPST16, GKP+13] that relying on LWE one can build randomized encoding schemes for Turing machines where the time of encoding only grows with the size of the output of the computation, rather than its running time. Thus one can simply encode the computation $\mathsf{PHFE.Enc}(\mathsf{P}, \mathsf{S})$ using the LWE based scheme and be done. In fact this was done in the work [JLS21a].

More so, up until now, this is the only way we know how to convert output size succinct computations into time succinct encodings. On the other hand, what we are after will also imply such a notion generically using the same transformation [LPST16]. Since we do not rely on LWE, this will actually give us a mechanism to do this task without LWE. Therefore, in this work we go back to basics and tackle this question directly.

High-level approach for PRE: Our high level approach is to actually boost a sublinear randomized encoding in NC^0. We take such an RE scheme, where Encode is a degree d computation for some constant d, and compile it to a PRE scheme where the encoding can be computed by degree $(O(d), 2)$ polynomials. Here is the approach:

- Since the public input P is supposed to hide $\mathbf{x}, \mathbf{r}$, we will set P as an encryption of $\mathbf{x}$ using a special purpose homomorphic encryption scheme, and,
- We set S to contain the secret key of this homomorphic encryption and some other "pre-processed information" about the encryption, so that $\mathsf{PRE.Encode}(C, (\mathsf{P}, \mathsf{S}))$ follows the following template: it first computes $\hat{\mathsf{P}}_C$ by performing "homomorphic evaluation" on P, where $\hat{\mathsf{P}}_C$ is an encryption of $\mathsf{RE.Encode}(C, \mathbf{x}, \mathbf{r})$. We will ensure that this homomorphic evaluation is a degree d operation (since $\mathsf{RE.Encode}$ is degree d). Finally, using S we can perform "decryption" of $\hat{\mathsf{P}}_C$ to derive $\mathsf{RE.Encode}(C, \mathbf{x}, \mathbf{r})$, where this decryption is a degree 1 polynomial in $\hat{\mathsf{P}}_C$ and degree two in S.
- We will ensure that P, S can be computed by a circuit of size sublinear in size of C.

To make things simpler and modular, let us recall the necessary structure we of the RE scheme we need for the rest of the overview. In an RE scheme, we can encode any input $\mathbf{x}$ and circuit C as $\mathsf{Encode}(C, \mathbf{x}, \mathbf{r}) = \mathbf{y}$. The circuit $C : \{0,1\}^n \to \{0,1\}^*$ that the scheme handles are of size $m = n^{1+\epsilon}$ for some $\epsilon > 0$. Further, the length of the encoding $\mathbf{y}$ is $\ell_{\mathsf{RE}} = O(m\,\mathsf{poly}(\lambda))$. Since each output bit of $\mathbf{y}$ is computable by an NC^0 circuit, this means that each output bit is computable by a linear combination of $t = O(\ell_{\mathsf{RE}})$ different monomials of degree at most d that may potentially depend on the description of C. We now introduce a new tool, which will assist us with converting such an RE scheme into a PRE scheme. This new tool will only deal with evaluating polynomials.

New Tool: Preprocessed Polynomial Encoding. Thus to advance towards the goal, we focus on the following simpler task of polynomial evaluation. Let us say that we have degree at most d monomials denoted by sets $Q_1, \ldots, Q_{m_{\mathsf{PPE}}}$ for $m_{\mathsf{PPE}} = n_{\mathsf{PPE}}^{1+\epsilon}$ for some $\epsilon > 0$ where each Q_i is a set of size d and defines a monomial $\mathsf{Mon}_{Q_i}(\mathbf{x}_{\mathsf{PPE}}) = \Pi_{j \in Q_i} x_{\mathsf{PPE},j}$. Such monomials, for example, could be the monomials involved in the $\mathsf{RE}.\mathsf{Encode}(C, \cdot)$ operation. Is it possible for one to compute a preprocessing of an input $\mathbf{x}_{\mathsf{PPE}} \in \{0,1\}^{n_{\mathsf{PPE}}}$ in time sublinear in m_{PPE}, a preprocessing (P, S) such that each $\mathsf{Mon}_{Q_i}(\mathbf{x}_{\mathsf{PPE}})$ can be computed by a degree $(O(d), 2)$ polynomial in (P, S). Further we would like to maintain that P hides $\mathbf{x}_{\mathsf{PPE}}$ as before?

This notion is referred to as a preprocessed polynomial encoding scheme PPE. Clearly such a notion seems useful here to bootstrap an RE scheme as an RE scheme also involves computing t degree d monomials. However, note that even if we can realize such a notion, we are still not done because in the RE scheme above the monomials $Q_1, \ldots, Q_{m_{\mathsf{PPE}}}$ used by $\mathsf{Encode}(C, \cdot)$ can depend on C and doesn't allow computing encodings for all circuits C. We handle this later, by constructing an RE scheme where the monomial set does not depend on C.

Constructing Preprocessed Polynomial Encoding. Let us now go back the question of constructing PPE (which is in turn useful to build PRE). As stated before, our approach is to simply encrypt $\mathbf{x}_{\mathsf{PPE}} \in \{0,1\}^{n_{\mathsf{PPE}}}$ using a suitable homomorphic encryption scheme. We actually encrypt it using LPN, thereby setting P as:

$$\mathsf{P} = \mathbf{A}, \mathbf{b} = \mathbf{s}\mathbf{A} + \mathbf{e} + \mathbf{x}_{\mathsf{PPE}} \mod p$$

where $\mathbf{A} \leftarrow \mathbb{Z}_p^{\dim \times n_{\mathsf{PPE}}}$, $\mathbf{s} \leftarrow \mathbb{Z}_p^{1\times \dim}$ and the dimension dim is set to be n_{PPE}^{ρ} for some constant $\rho \ll \epsilon$. The errors $\mathbf{e}$ is chosen so that each coordinate is non-zero with probability $\dim^{-\delta}$ for $\delta > 0$ associated with the LPN assumption.

Observe that P can be computed in time $\widetilde{O}(n_{\mathsf{PPE}} \cdot \dim^2)$ which is sublinear in m_{PPE} as $\dim \ll n_{\mathsf{PPE}}^{\epsilon}$. Thus the remaining goal is to come up with S in time sublinear in the number of monomials m_{PPE} such that S can be used to evaluate $\mathsf{Mon}_{Q_i}(\mathbf{x}_{\mathsf{PPE}})$ for all $i \in [m_{\mathsf{PPE}}]$.

PPE: **First Attempt.** A first attempt is to set $\mathsf{S} = (\mathbf{s}||1)^{\otimes \lceil \frac{d}{2} \rceil}$. Note that this fine time-wise because it can be done in time $\widetilde{O}(\dim^{\lceil \frac{d}{2} \rceil})$ which is allowed. The reason of this choice is because the following equation holds:

$$\mathsf{Mon}_{Q_i}(\mathbf{b} - \mathbf{A}\mathbf{s}) = \mathsf{Mon}_{Q_i}(\mathbf{x}_{\mathsf{PPE}} + \mathbf{e}).$$

Our main observations are the following: i) the above quantity can be computed by a polynomial that is degree d in $(\mathbf{A}, \mathbf{b})$ and degree two in $\mathsf{S} = (\mathbf{s}||1)^{\otimes \lceil \frac{d}{2} \rceil}$ and, ii) since the error $\mathbf{e}$ is sparse, and Q_i depends on only a constant number d variables, and thus with probability $1 - O(\dim^{-\delta})$, $\mathsf{Mon}_{Q_i}(\mathbf{x}_{\mathsf{PPE}} + \mathbf{e}) = \mathsf{Mon}_{Q_i}(\mathbf{x}_{\mathsf{PPE}})$.

This is nice because the number of monomials Q_i such that $\mathsf{Mon}_{Q_i}(\mathbf{x}_{\mathsf{PPE}}+\mathbf{e}) \neq \mathsf{Mon}_{Q_i}(\mathbf{x}_{\mathsf{PPE}})$ in expectation is $O(m_{\mathsf{PPE}} \cdot \dim^{-\delta})$ which is sublinear in m_{PPE}! This suggests the natural strategy: we will come up with another component S_1 that is computable in sublinear time such that it compresses the sparse correction vector $\mathsf{Corr} = \{\mathsf{Mon}_{Q_i}(\mathbf{x}_{\mathsf{PPE}}) - \mathsf{Mon}_{Q_i}(\mathbf{x}_{\mathsf{PPE}} + \mathbf{e})\}_{i \in [m_{\mathsf{PPE}}]}$. The compression should be such that decompression should be possible using a quadratic polynomial in S_1. To be precise we will make sure that for every $i \in [m_{\mathsf{PPE}}]$, $g_i(\mathsf{S}_1) = \mathsf{Corr}[i]$ using a degree two polynomial g_i. This can now be added with $\mathsf{Mon}_{Q_i}(\mathbf{b}-\mathbf{sA})$ to get the required output. We will go over the details of this idea in a little while, but before we do that we must point out an important issue with this approach.

Unfortunately, this approach will run into the following counting argument: The LPN samples $\mathbf{A}, \mathbf{sA} + \mathbf{e} + \mathbf{x}_{\mathsf{PPE}}$ losses information of about $\Omega(n_{\mathsf{PPE}} \dim^{-\delta})$ entries of $\mathbf{x}_{\mathsf{PPE}}$ *even given* $\mathbf{s}$. Since the set of monomials $\mathcal{Q} = \{Q_1, \ldots, Q_{m_{\mathsf{PPE}}}\}$ are arbitrary, the circuit computing the preprocessing must at least store the description of $\mathcal{Q}$ in order to compute and compress the monomials that need correction. This suggests that the size of the circuit preprcessing must grow linearly with m_{PPE}. This is not just a hurdle just for this particular strategy of setting S, in fact, must apply to any strategy that sets P as $(\mathbf{A}, \mathbf{b})$. In order to get around this hurdle, our main idea is to solve this using amortization.

Amortization. To get around the hurdle pointed above, we ask a somewhat simpler question: the question is if we can "batch-preprocess" in sublinear time? To make it precise, say we have k_{PPE} input vectors $\mathbf{x}_1 \ldots, \mathbf{x}_{k_{\mathsf{PPE}}}$ each of dimension n_{PPE}, and we are interested in learning linear combinations of $\{\mathsf{Mon}_{Q_i}(\mathbf{x}_j)\}_{i \in [m_{\mathsf{PPE}}], j \in [k_{\mathsf{PPE}}]}$ where $\mathcal{Q} = \{Q_1, \ldots, Q_{m_{\mathsf{PPE}}}\}$ are degree d monomials of interest. Then the question is, can we process $\{\mathbf{x}_1, \ldots, \mathbf{x}_{k_{\mathsf{PPE}}}\}$ into a public and a secret input (P, S) in time sublinear in $m_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}$ (which is the number of monomials of interest), such that each $\mathsf{Mon}_{Q_i}(\mathbf{x}_j) = g_{i,j}(\mathsf{P}, \mathsf{S})$ for some degree $(O(d), 2)$ polynomial $g_{i,j}$?

The hope is that even if we can't win when $k_{\mathsf{PPE}} = 1$, we may be able to win overall when k_{PPE} is large. What does sublinear in $m_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}$ mean? Precisely, it means that the time to compute grows as $T_{\mathsf{PPE}} = \widetilde{O}(m_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}^{1-c_1} + (m_{\mathsf{PPE}}^{1-c_2} + n_{\mathsf{PPE}}) k_{\mathsf{PPE}}^{c_2})$. This is because when $m_{\mathsf{PPE}} = n_{\mathsf{PPE}}^{1+\epsilon}$, then we can choose $k_{\mathsf{PPE}} = n_{\mathsf{PPE}}^{\Omega(1)}$ such that $T_{\mathsf{PPE}} = \widetilde{O}((m_{\mathsf{PPE}} k_{\mathsf{PPE}})^{1-\gamma})$ for some constant $\gamma > 0$. This gives rise to two question:

- Can we construct PPE?

- Is this amortization useful to construct PRE?

We answer the second question first.

Constructing PRE using (amortized) PPE**.** Recall that in order to construct PRE scheme, we need a randomized encoding scheme that is compatible with our PPE scheme. We observe that relying on decomposability properties of Yao's garbling scheme [Yao86] based on PRG's in NC^0, and inspired from prior work [Lin16, LV16, Lin17, AS17, LT17] we can exactly construct such a scheme. We will call such a scheme an amortized randomized encoding scheme or ARE. The basic idea behind this is the following. Consider the circuits $C : \{0,1\}^{n_{\mathsf{ARE}}} \to \{0,1\}^{m_{\mathsf{ARE}} k_{\mathsf{ARE}}}$ where every output bit is computable by a circuit of fixed size say λ. We first observe that, constructing a PRE scheme for this class is also enough to construct $i\mathcal{O}$ for all circuits. Therefore we will construct an ARE scheme for the same class. Now consider our candidate ARE scheme where the encoding operation is defined as:

$$\mathsf{ARE.Encode}(C, \mathbf{x}, \mathbf{r}_1, \ldots, \mathbf{r}_{k_{\mathsf{ARE}}}) = \mathsf{Yao.Gb}(C_1, \mathbf{x}, \mathbf{r}_1), \ldots, \mathsf{Yao.Gb}(C_{k'}, \mathbf{x}, \mathbf{r}_{k_{\mathsf{ARE}}}),$$

where $C_1, \ldots, C_{k_{\mathsf{ARE}}}$ are sub-circuits such that C_i computes i^{th} block of outputs of circuit C of size m_{ARE} and $\mathsf{Yao.Gb}$ expands $\mathbf{r} \in \{0,1\}^{n_{\mathsf{ARE}}}$ using a PRG in NC^0 and uses that randomness to garble the circuit using the Yao's garbling scheme. Observe that in this case, the output length of each block $\mathsf{Yao.Gb}(C_i, \mathbf{x}, \mathbf{r}_i)$ is $\widetilde{O}(n_{\mathsf{ARE}}+m_{\mathsf{ARE}})$. Since the garbling is computed by an NC^0 circuit, the number of the monomials involved to compute $\mathsf{Yao.Gb}(C_i, \mathbf{x}, \mathbf{r}_i)$ are $\widetilde{O}(n_{\mathsf{ARE}}+m_{\mathsf{ARE}})$. This almost solves the problem, except the monomials used to compute $\mathsf{Yao.Gb}(C_i, \cdot)$ for each block depends on circuits C_i. We would like each block to be computable using same set of monomials irrespective of the circuit C_i. We present a simple solution.

$$\mathsf{ARE.Encode}(C, \mathbf{x}, \mathbf{r}_1, \ldots, \mathbf{r}_{k_{\mathsf{ARE}}}) = \mathsf{Yao.Gb}(U, (C_1, \mathbf{x}), \mathbf{r}_1), \ldots, \mathsf{Yao.Gb}(U(C_{k_{\mathsf{ARE}}}, \mathbf{x}), \mathbf{r}_{k_{\mathsf{ARE}}}),$$

where U is a universal circuit that takes as input $C_i, \mathbf{x}$ and outputs $C_i(\mathbf{x})$. Universal circuits for emulating such computations are known be of size $\widetilde{O}(n_{\mathsf{ARE}}+m_{\mathsf{ARE}})$ [Val76]. This indeed gives rise to a set of $t_{\mathsf{ARE}} = \widetilde{O}(n_{\mathsf{ARE}} + m_{\mathsf{ARE}})$ monomials $Q_1, \ldots, Q_{t_{\mathsf{ARE}}}$ such that each $\mathsf{Yao.Gb}(U, (C_i, \mathbf{x}), \mathbf{r}_i)$ when seen as a polynomial is a linear combination of $\{\mathsf{Mon}_{Q_j}(\mathbf{x}, \mathbf{r}_i)\}_{j \in [t_{\mathsf{ARE}}]}$. This is shown by closely inspecting the construction of garbled circuits. We show this in Section 3.1.2.

Constructing PPE. Coming back to the construction of PPE, we now want to batch preprocess $\mathbf{x}_1, \ldots, \mathbf{x}_{k_{\mathsf{PPE}}}$ each in $\{0,1\}^{n_{\mathsf{PPE}}}$ into a preprocessed input (P, S) such that we can compute $\{\mathsf{Mon}_{Q_i}(\mathbf{x}_j)\}_{i \in [m_{\mathsf{PPE}}], j \in [k_{\mathsf{PPE}}]}$ using degree $(O(d), 2)$ polynomials in (P, S). Further it is requied that P hides $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_{k_{\mathsf{PPE}}})$ and the time to compute (P, S) is sublinear in $m_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}$. Our approach is same as before. We set:

$$\mathsf{P} = \{\mathbf{A}_j, \mathbf{b}_j = \mathbf{s}\mathbf{A}_j + \mathbf{e}_j + \mathbf{x}_j\}_{j \in [k_{\mathsf{PPE}}]},$$

where $\mathbf{A}_j \leftarrow \mathbb{Z}_p^{n_{\mathsf{PPE}} \times k_{\mathsf{PPE}}}$, $\mathbf{s} \leftarrow \mathbb{Z}_p^{1 \times k_{\mathsf{PPE}}}$ and $\mathbf{e}_j \in \mathbb{Z}_p^{k_{\mathsf{PPE}}}$ where each coordinate is zero with probability $k_{\mathsf{PPE}}^{-\delta}$. Notice that the dimension is set to be k_{PPE}. This is okay because k_{PPE} is polynomially related to n_{PPE}.

Now we focus on how to compute S. Our high level strategy is to compute $\mathsf{S} = (\mathsf{S}_0, \mathsf{S}_1, \ldots, \mathsf{S}_{m_{\mathsf{PPE}}})$ where each S_i for $i \in [m_{\mathsf{PPE}}]$ compresses corrections for the computations corresponding to set Q_i: $\{\mathsf{Mon}_{Q_i}(\mathbf{x}_j)\}_{j \in [k_{\mathsf{PPE}}]}$. We will show that each S_i can be computed in (amortized) time roughly $\widetilde{O}(k_{\mathsf{PPE}}^{1-\Omega(1)})$. Finally, we will set S_0 as before to $(1||\mathbf{s})^{\otimes \frac{d}{2}}$. We now describe how each S_i is formed, and then describe the intuition why it can be computed efficiently. This will complete the overview because P can be computed in time $\widetilde{O}(n_{\mathsf{PPE}} k_{\mathsf{PPE}}^2)$ and S_0 in time $\widetilde{O}(k_{\mathsf{PPE}}^{d/2})$.

Computing S_i. As a first step, we compute Corr_i as an array of size k_{PPE}, where $\mathsf{Corr}_i[j] = \mathsf{Mon}_{Q_i}(\mathbf{x}_j) - \mathsf{Mon}_{Q_i}(\mathbf{x}_j + \mathbf{e}_j)$ for $j \in [k_{\mathsf{PPE}}]$. Now observe that as before Corr_i is a sparse array with $O(k_{\mathsf{PPE}}^{1-\delta})$ non-zero elements with overwhelming probability. We exploit this, to compute S_i.

We form $t_1 = k_{\mathsf{PPE}}^{1-\delta}$ matrices $\{\mathbf{M}_{i,\gamma}\}_{\gamma\in[t_1]}$ of size $T \times T$ such that the parameters satisy $T^2 \cdot t_1 = k_{\mathsf{PPE}}$. We will then simply arrange Corr_i bijectively into k_{PPE} spots inside t_1 matrices. To do so, let ϕ be an arbitrary bijective map that maps $[k_{\mathsf{PPE}}]$ into these matrices. This can be done by sampling $\phi = (\phi_{bkt}, \phi_{ind})$ where $\phi_{bkt} : [k_{\mathsf{PPE}}] \to [t_1]$ assigns every element into the corresponding matrix, and $\phi_{ind} : [k_{\mathsf{PPE}}] \to [T] \times [T]$ gives the location in the corrsponding matrix. Such a function can be arbitrarily chosen[2].

Once we have such a function ϕ, one can set: $\mathsf{Corr}_i[j] = \mathbf{M}_{i,j_1}[j_2, j_3]$ where $\phi_{bkt}(j) = j_1$ and $\phi_{ind}(j) = (j_2, j_3)$. The point of doing this is that since the number non-zero entries in Corr_i is $O(t_1)$, in expectation, each matrix $\mathbf{M}_{i,\gamma}$ has a constant number of non-zero elements. Using concentration bounds, we can in fact show that with overwhelming probability in λ each matrix has at most λ non-zero entries. Thus, its rank is less than or equal to λ and we can compute $\mathbf{U}_{i,\gamma}, \mathbf{V}_{i,\gamma} \in \mathbb{Z}_p^{T\times\lambda}$ such that $\mathbf{M}_{i,\gamma} = \mathbf{U}_{i,\gamma} \cdot \mathbf{V}_{i,\gamma}^{\top}$. We set $\mathsf{S}_i = \{\mathbf{U}_{i,\gamma}, \mathbf{V}_{i,\gamma}\}_{\gamma\in[t_1]}$. Observe that the size S_i is $\widetilde{O}(k_{\mathsf{PPE}}^{1-\delta}T) = \widetilde{O}(k_{\mathsf{PPE}}^{1-\frac{\delta}{2}})$ as $T = k_{\mathsf{PPE}}^{\frac{\delta}{2}}$, which is what we wanted. We now check that for this setting of S_i, we can compute each $\mathsf{Mon}_{Q_i}(\mathbf{x}_j)$ using a degree $(d, 2)$ polynomial in P, S:

$$\begin{aligned}
&\mathsf{Mon}_{Q_i}(\mathbf{b}_j - \mathbf{s}\mathbf{A}_j) + \mathbf{U}_{i,j_1} \cdot \mathbf{V}_{i,j_1}^{\top}[j_2, j_3],\\
=&\mathsf{Mon}_{Q_i}(\mathbf{b}_j - \mathbf{s}\mathbf{A}_j) + \mathbf{M}_{i,j_1}[j_2, j_3],\\
=&\mathsf{Mon}_{Q_i}(\mathbf{x}_j + \mathbf{e}_j) + \mathsf{Corr}_i[j],\\
=&\mathsf{Mon}_{Q_i}(\mathbf{x}_j + \mathbf{e}_j) + \mathsf{Mon}_{Q_i}(\mathbf{x}_j) - \mathsf{Mon}_{Q_i}(\mathbf{x}_j + \mathbf{e}_j),\\
=&\mathsf{Mon}_{Q_i}(\mathbf{x}_j),
\end{aligned}$$

where $\phi(j) = (j_1, j_2, j_3)$.

[2]One such function ϕ can be computed by dividing j by t_1 first and setting the remainder to $\phi_{bkt}(j) = j_1$. Then the remainder of this division is further divided by T. The quotient and remainder of this division can be set to $\phi_{ind}(j) = (j_2, j_3)$.

Computability of S_i in sublinear time. Above, we showed that the size of S_i is $\widetilde{O}(k_{\mathsf{PPE}}^{1-\delta/2})$. However, this does not mean that it can be computed by a circuit of small size. We now argue why $\mathsf{S}_1, \ldots, \mathsf{S}_{m_{\mathsf{PPE}}}$ can actually be computed by a circuit of size $\widetilde{O}(n_{\mathsf{PPE}} k_{\mathsf{PPE}}^2 + m_{\mathsf{PPE}} k_{\mathsf{PPE}}^{1-\delta/2})$. We break down the task of computing $\mathsf{S}_1, \ldots, \mathsf{S}_{m_{\mathsf{PPE}}}$ in two steps.

1. Clearly, to compute each S_i in amortized sublinear time in k_{PPE}, we cannot afford to compute the entire row Corr_i which has dimension k_{PPE}. Instead, we compute the list NZCorr_i of non-zero entries in Corr_i only, which has size $O(k_{\mathsf{PPE}}^{1-\delta})$. More precisely, NZCorr_i consists of tuples of the form

$$\mathsf{NZCorr}_i = \{(j,\ \phi(j) = (j_1, j_2, j_3),\ \mathsf{Corr}[i,j]) \mid j \in [k_{\mathsf{PPE}}],\ \mathsf{Corr}_i[j] \neq 0\} \ .$$

That is, it contains the index j of the non-zero entries in Corr_i, the matrix location they are assigned to $\mathbf{M}_{i,j_1}[j_2, j_2]$, and the value of the error $\mathsf{Corr}[i,j]$. Moreover, the list is sorted in ascending order with respect to coordinate j_1, so that tuples with the same value j_1 appear contiguously.

2. In the second step, we use these special lists $\{\mathsf{NZCorr}_i\}$ to compute S_i.

Let's see how to do each step in amortized sublinear time, starting with the easier second step.

<u>THE SECOND STEP:</u> Given NZCorr_i, we can compute S_i in time $\mathsf{poly}(\lambda)(k_{\mathsf{PPE}}^{1-\delta/2})$. This is done by making a single pass on NZCorr_i and generating rows and columns of $\{\mathbf{U}_{i,\gamma}, \mathbf{V}_{i,\gamma}\}_{\gamma \in [T}$ "on the fly". We can start by initializing these matrices with zero entries. Then for the ℓ'th tuple $(j, \phi(j) = (j_1, j_2, j_3), \mathsf{Corr}[i,j])$ in NZCorr_i, we set $\mathbf{U}_{i,j_1}[j_2, \ell] = \mathsf{Corr}[i,j]$ and $\mathbf{V}_{i,j_1}[j_3, \ell] = 1$. Since each matrix $\mathbf{M}_{i,\gamma}$ gets assigned at most λ non-zero entries, the index ℓ ranges from 1 up to λ, fitting the dimension of $\mathbf{U}$'s, and $\mathbf{V}$'s. Hence, this way of generating $\mathbf{U}_{i,\gamma}$ and $\mathbf{V}_{i,\gamma}$ guarantees that $\mathbf{M}_{i,\gamma} = \mathbf{U}_{i,\gamma}\mathbf{V}_{i,\gamma}^{\top}$.

<u>THE FIRST STEP:</u> Next, we first illustrate how to generate all lists $\{\mathsf{NZCorr}_i\}_{i \in [m_{\mathsf{PPE}}]}$ in sublinear time in $m_{\mathsf{PPE}} k_{\mathsf{PPE}}$, in the Random Access Memory (RAM) model. The first sub-

step is collecting information related to all the non-zero elements in the LPN errors $\{\mathbf{e}_j\}_{j\in[k_{\mathsf{PPE}}]}$ used to encrypt the inputs $\{\mathbf{x}_j\}_{j\in[k_{\mathsf{PPE}}]}$. More precisely, for every coordinate $l \in [n]$ in an input, form the list

$$\mathsf{NZInp}_l = \{(j, x_{j,l}, e_{j,l}) \mid e_{j,l} \neq 0\}_{j\in[k_{\mathsf{PPE}}]} \ .$$

That is, NZInp_l contains the index j of each input $\mathbf{x}_j$, such that, the l'th element $x_{j,l}$ is blinded by a non-zero error $e_{j,l} \neq 0$, as well as the values $x_{j,l}, e_{j,l}$ of the input and error elements. Tuples in this list are sorted in ascending order with respect to coordinate j. Note that these lists can be computed in time $O(n_{\mathsf{PPE}}k_{\mathsf{PPE}})$.

Now, think of a database that contains all $\{\mathsf{NZInp}_l\}_l$ and inputs $\{\mathbf{x}_j\}_j$, which can be randomly accessed. The second sub-step makes a pass over all monomials $Q_1, \ldots Q_{m_{\mathsf{PPE}}}$. Each monomial Q_i depends on at most d variables (out of n_{PPE} variables), say Q_i depends on variables at coordinates $\{l_1, \ldots, l_d\}$. For every monomial Q_i, with random access to the database, make a single pass on lists $\mathsf{NZInp}_{l_1}, \ldots, \mathsf{NZInp}_{l_d}$ and generate NZCorr_i on the fly. The fact that every list NZInp_l is sorted according to j ensures that the time spent for each Q_i is $O(k_{\mathsf{PPE}}^{1-\delta})$. Thus, in the RAM model $\{\mathsf{NZCorr}_i\}_i$ can be constructed in sublinear time $O(m_{\mathsf{PPE}}k_{\mathsf{PPE}}^{1-\delta})$. All we need to do now is coming up with a circuit to do the same.

<u>Circuit Conversion:</u> To obtain such a circuit, we examine each and every step inside the above RAM program and then replace them by suitable (sub)circuits, while preserving the overall running-time. Since the conversion is very technical, we refer the reader to Chapter 4 for details, and only highlight some of the tools used in the conversion. We make extensive use of sorting circuits of almost linear size [AKS83] and Turing machine to circuit conversions. For example, at some point we have to replace RAM memory lookups by circuits. To do so, we prove the following simple lemma about RAM look up programs. A RAM lookup program $P^{\mathsf{lookup}}_{q,N}$ indexed with a number $N \in \mathbb{N}$ and a number $q \in \mathbb{N}$ is a program with the following structure: It takes as input q indices $\{i_1, \ldots, i_q\}$ and a database $\mathsf{DB} \in \{0,1\}^N$ and it outputs $\{\mathsf{DB}[i_1], \ldots, \mathsf{DB}[i_q]\}$. We show that this can be implemented efficiently by a circuit:

Lemma 2.2.1. *Let $q, N \in \mathbb{N}$. A RAM lookup program $P^{\mathsf{RAM}}_{q,N}$ (that looks up q indices from a database of size N) can be implemented by an efficiently uniformly generatable boolean circuit of size $O((q+N)\,\mathsf{poly}(\log_2(q \cdot N)))$ for some polynomial poly.*

Please see Section 4 for how we use the above lemma and other technical details.

2.2.1.1 Outline

This completes are technical overview. In the rest of the chapter, we formally define Functional Encryption in Section 2.3. In Section 2.4 we define the notion of a Partially Hiding Functional Encryption scheme. In Section 2.5 we define the notion of our PRE scheme. Finally in Section 2.6 we show how to construct a sublinear functional encryption and $i\mathcal{O}$ from these two primitives. We discuss the constructions of PHFE and PRE in later chapters.

2.3 Functional Encryption Definition

We denote by $\mathcal{F}_{\mathsf{FE}} = \{\mathcal{F}_{\mathsf{FE},n_{\mathsf{FE}},m_{\mathsf{FE}},\lambda}\}_{n_{\mathsf{FE}} \in \mathsf{poly}, m_{\mathsf{FE}} \in \mathsf{poly}, \lambda \in \mathbb{N}}$ an abstract function class, which is parameterized by security parameter $\lambda \in \mathbb{N}$ and polynomials $n_{\mathsf{FE}}(\cdot)$, $m_{\mathsf{FE}}(\cdot)$. This class consists of all boolean circuits with $n_{\mathsf{FE}} = n_{\mathsf{FE}}(\lambda)$ input bits, $m_{\mathsf{FE}} = m_{\mathsf{FE}}(\lambda)$ output bits, and where every output bit can be computed by a circuit of size λ. This is the class of circuit for which we will construct a functional encryption.

We now define the syntax of the functional encryption scheme.

Definition 2.3.1. *(Syntax of a FE Scheme.) A functional encryption scheme FE for the function class $\mathcal{F}_{\mathsf{FE},n_{\mathsf{FE}},m_{\mathsf{FE}},\lambda}$ consists of the following PPT algorithms:*

- $\mathsf{Setup}(1^\lambda, 1^{n_{\mathsf{FE}}}, 1^{m_{\mathsf{FE}}})$*: On input the security parameter λ, parameters $n_{\mathsf{FE}}(\lambda)$ and $m_{\mathsf{FE}}(\lambda)$, it outputs a public key and a master secret key pair* $(\mathsf{PK}, \mathsf{MSK})$.
- $\mathsf{Enc}(\mathsf{PK}, \mathbf{x})$*: Given as input the public key PK and a message $\mathbf{x} \in \{0,1\}^{n_{\mathsf{FE}}(\lambda)}$, it outputs*

a ciphertext CT.

- $\mathsf{KeyGen}(\mathsf{MSK}, f)$*: Given as input the master secret key* MSK *and a function* $f \in \mathcal{F}_{\mathsf{FE},\lambda,n_{\mathsf{FE}},m_{\mathsf{FE}}}$*, it outputs a functional decryption key* SK_f.

- $\mathsf{Dec}(\mathsf{SK}_f, \mathsf{CT})$*: Given a functional decryption key* SK_f *and a ciphertext* CT*, it deterministically outputs a value* $\mathbf{y}$ *in* $\{0,1\}^{m_{\mathsf{FE}}(\lambda)}$*, or* $\perp$ *if it fails.*

Remark 2.3.1 (On secret-key schemes)**.** One can also consider a secret key functional encryption scheme, where the encryption algorithm must use MSK to encrypt a message. Such FE schemes also imply $i\mathcal{O}$ [BNPW16, KNT18]. However since we directly build a public key encryption scheme, we do not discuss about secret-key schemes in this thesis.

We now define the correctness of decryption property.

Definition 2.3.2. *(Correctness.) An FE scheme* FE *for the functionality* $\mathcal{F}_{\mathsf{FE},\lambda,n_{\mathsf{FE}},m_{\mathsf{FE}}}$ *is correct if for any polynomials* $n_{\mathsf{FE}}, m_{\mathsf{FE}} : \mathbb{N} \to \mathbb{N}$ *any security parameter* $\lambda \in \mathbb{N}$*, any* $\mathbf{x} \in \{0,1\}^{n_{\mathsf{FE}}(\lambda)}$*, and every function* $f \in \mathcal{F}_{\mathsf{FE},n_{\mathsf{FE}},m_{\mathsf{FE}},\lambda}$ *we have:*

$$\Pr\begin{bmatrix} (\mathsf{PK}, \mathsf{MSK}) \leftarrow \mathsf{Setup}(1^\lambda, 1^{n_{\mathsf{FE}}(\lambda)}, 1^{m_{\mathsf{FE}}(\lambda)}) \\ \mathsf{CT} \leftarrow \mathsf{Enc}(\mathsf{PK}, \mathbf{x}) \\ \mathsf{SK}_f \leftarrow \mathsf{KeyGen}(\mathsf{SK}, f) \\ \mathsf{Dec}(\mathsf{SK}_f, \mathsf{CT})) = f(\mathbf{x}) \end{bmatrix} = 1.$$

We now give the security definition for such a functional encryption scheme.

Definition 2.3.3 (IND security)**.** *We say an FE scheme* FE *for functionality* $\mathcal{F}_{\mathsf{FE},\lambda,n_{\mathsf{FE}}(\cdot),m_{\mathsf{FE}}(\cdot)}$ *is IND secure if for all stateful PPT adversaries* $\mathcal{A}$*, there exists a negligible function* negl *such that , we have:*

$$\mathsf{Adv}^{\mathsf{IND}}_{\mathsf{FE},\mathcal{A}}(\lambda) := 2 \cdot |1/2 - \Pr[1 \leftarrow \mathsf{IND}^{\mathsf{FE}}_{\mathcal{A}}(1^\lambda)]| < \mathsf{negl}(\lambda),$$

where the experiment $\mathsf{IND}^{\mathsf{FE}}_{\mathcal{A}}(1^\lambda)$ *is defined below.*

```
$\underline{\mathsf{IND}^{\mathsf{FE}}_{\mathcal{A}}(1^\lambda)}$:
$(1^{n_{\mathsf{FE}}}, 1^{m_{\mathsf{FE}}}) \leftarrow \mathcal{A}(1^\lambda)$
$\{\mathbf{x}_i \in \{0,1\}^{n_{\mathsf{FE}}}\}_{i\in\{0,1\}}$ , $\{f_j \in \mathcal{F}_{\mathsf{FE},\lambda,n_{\mathsf{FE}},m_{\mathsf{FE}}}\}_{j\in Q_{\mathsf{SK}}} \leftarrow \mathcal{A}$
$(\mathsf{PK}, \mathsf{MSK}) \leftarrow \mathsf{Setup}(1^\lambda, 1^{n_{\mathsf{FE}}}, 1^{m_{\mathsf{FE}}})$, $b \leftarrow \{0,1\}$
$\mathsf{CT} \leftarrow \mathsf{FE.Enc}(\mathsf{PK}, \mathbf{x}_b)$, $\forall j \in [Q_{\mathsf{SK}}] : \mathsf{SK}_j \leftarrow \mathsf{KeyGen}(\mathsf{MSK}, f_j)$
$b' \leftarrow \mathcal{A}(\mathsf{PK}, \mathsf{CT}, \{\mathsf{SK}_j\}_{j\in Q_{\mathsf{SK}}})$
Return $1$ if $b = b'$ and $\forall\, j \in [Q_{\mathsf{SK}}], f_j(\mathbf{x}_0) = f_j(\mathbf{x}_1)$, $0$ otherwise.
```

Further, we say that FE *satisfies subexponential security if* $\mathsf{negl}(\lambda) = 2^{-\lambda^{\Omega(1)}}$.

Remark 2.3.2 (On number of key queries)**.** For this work we concern with the case when the number of function key queries $Q_{\mathsf{SK}} = 1$. This is because, as shown in [AJ15, BV15], such an FE scheme, additionally satisfying sublinear encryption time implies $i\mathcal{O}$ under subexponential security loss. Under polynomial security loss, such an FE scheme also implies an FE scheme where Q_{SK} is arbitrary polynomial. This was shown in the works of [GS16, LM16].

We finally describe the property of sublinear encryption time. This property will be referred to as "sublinearity".

Definition 2.3.4 (Sublinearity)**.** *Let* FE *be an FE scheme for the functionality* $\mathcal{F}_{\mathsf{FE},\lambda,n_{\mathsf{FE}},m_{\mathsf{FE}}}$*. We say that* FE *satisfies sublinear encryption time property (or simply sublinearity), if there exists a constant* $\epsilon \in (0,1)$ *and a polynomial* poly *such that for any polynomials* $n_{\mathsf{FE}}, m_{\mathsf{FE}}$ *and any security parameter* λ*, all* PK *in the support of* $\mathsf{Setup}(1^\lambda, n_{\mathsf{FE}}(\lambda), m_{\mathsf{FE}})$ *the size of the circuit computing* $\mathsf{FE.Enc}(\mathsf{PK}, \cdot)$ *is* $O((n_{\mathsf{FE}} + m_{\mathsf{FE}}^{1-\epsilon})\,\mathsf{poly}(\lambda))$*.*

Remark 2.3.3 (Sublinear Functional Encryption for Polynomial Sized Circuits)**.** We could have defined the above notion for a circuit class $\mathcal{F}_{\mathsf{P},\lambda,n_{\mathsf{FE}},m_{\mathsf{FE}}}$ consisting of all circuits with input length $n_{\mathsf{FE}}(\lambda)$ and size $m_{\mathsf{FE}}(\lambda)$ (as opposed to the number of outputs) without having any restriction about the size of the circuit computing each output bit. A functional encryption scheme for this class will be referred to as a functional encryption scheme for all circuits.

The notion of sublinearity is then defined by requiring the size of the encryption circuit to be $O((n_{\mathsf{FE}}+m_{\mathsf{FE}}^{1-\epsilon})\,\mathsf{poly}(\lambda))$ for some $\epsilon > 0$. It was shown in [AJS15] that using a straightforward application of decomposable Randomized Encoding (such as Yao's garbled circuits [Yao86]), any sublinear functional encryption scheme for $\mathcal{F}_{\mathsf{FE},\lambda,n_{\mathsf{FE}},m_{\mathsf{FE}}}$ can be converted to a sublinear functional encryption for all circuits. We choose to work with the class $\mathcal{F}_{\mathsf{FE},\lambda,n_{\mathsf{FE}},m_{\mathsf{FE}}}$ because this class is better compatible with the notion of PRE.

2.3.1 Bootstrapping Theorems for Functional Encryption to $i\mathcal{O}$

In this section, we briefly survey theorems from the literature that prove that a sublinear functional encryption implies $i\mathcal{O}$. We rely on these results for a construction of an $i\mathcal{O}$ scheme. The first result in this line showed:

Theorem 2.3.1 ([AJ15, BV15]). *If there exists a subexponentially secure public key sublinear functional encryption for all polynomial size circuits, then there exists an indistinguishability obfuscation scheme.*

Further, the result above is constructive and gives an actual construction of $i\mathcal{O}$ starting from such a functional encryption scheme. Building upon these works, there have been several other results (such as [BNPW16, KNT18]) studying equivalence from various other kinds of functional encryption such as secret key functional encryption to $i\mathcal{O}$. We construct a public key sublinear functional encryption scheme for $\mathcal{F}_{\mathsf{FE},n_{\mathsf{FE}}(\lambda),m_{\mathsf{FE}}(\lambda),\lambda}$, where n_{FE} and m_{FE} are arbitrary polynomials, which consists of all circuits with $n_{\mathsf{FE}}(\lambda)$ input bits and $m_{\mathsf{FE}}(\lambda)$ output bits where every output bit is computed by a circuit of size λ. It was shown in [AJS15] that, a sublinear functional encryption for this class implies sublinear functional encryption for circuits. Namely:

Theorem 2.3.2 ([AJS15]). *Assuming there exists a public key sublinear functional encryption for $\{\mathcal{F}_{\mathsf{FE},n_{\mathsf{FE}},n_{\mathsf{FE}}^{1+\epsilon},\lambda}\}_{n_{\mathsf{FE}}\in\mathsf{poly},\lambda\in\mathbb{N}}$ for some constant $\epsilon > 0$, there exists a public key sublinear functional encryption scheme for all circuits.*

Thus, from the results above we get:

Theorem 2.3.3 ([AJ15, BV15, AJS15]). *If there exists a constant $\epsilon > 0$ such that there exists a subexponentially secure public key sublinear functional encryption for $\mathcal{F}_{\mathsf{FE}} = \{\mathcal{F}_{\mathsf{FE},\lambda,n_{\mathsf{FE}},m_{\mathsf{FE}}=n_{\mathsf{FE}}^{1+\epsilon}}\}$, then there exists an indistinguishability obfuscation scheme for all circuits.*

2.4 Ingredient 1: Partially Hiding Functional Encryption

We now give the formal definition of a PHFE scheme. In a nutshell, syntactically, it is a generalization of a functional encryption. As the name suggests a PHFE scheme has the ability to hide the input "partially". The input has two components. A public input P and a secret input S. Any decryptor that has a function key for a function f, can learn P along with the value $f(\mathsf{P},\mathsf{S})$. Therefore a PHFE scheme for general circuits also implies a functional encryption scheme for circuits by simply setting P to $\perp$. We consider PHFE for the following function class:

Function class $\mathcal{F}_{\mathsf{PHFE}}$: The function class $\mathcal{F}_{\mathsf{PHFE}} = \{\mathcal{F}_{\mathsf{PHFE},d,p,n_{\mathsf{PHFE}}}\}_{d\in\mathbb{N},\ p\in\mathsf{PRIMES},n_{\mathsf{PHFE}}\in\mathbb{N}}$ is indexed by a degree $d \in \mathbb{N}$, a modulus p which is a prime, and a parameter $n_{\mathsf{PHFE}} \in \mathbb{N}$. The class consists of all polynomials f that takes as input two vectors $\mathsf{P},\mathsf{S} \in \mathbb{Z}_p^{n_{\mathsf{PHFE}}}$ and has the following form:

$$f(\mathsf{P},\mathsf{S}) = \sum_{j,k} f_{j,k}(\mathsf{P}) \cdot \mathsf{S}_j \cdot \mathsf{S}_k \mod p,$$

where every $f_{j,k}(\mathsf{P})$ is at most degree d polynomial over $\mathbb{Z}_p$.

Definition 2.4.1. *(Syntax of a PHFE Scheme.) A public key partially hiding functional encryption scheme,* PHFE*, for the functionality $\mathcal{F}_{\mathsf{PHFE}}$ consists of the following polynomial time algorithms:*

- $\mathsf{PPGen}(1^\lambda)$: *The public parameter generation algorithm is a randomized algorithm that* *takes as input a security parameter λ and outputs a string* $\mathsf{PP} = (\mathsf{crs}, p)$ *which consists of a modulus* p.

- $\mathsf{Setup}(d, 1^{n_{\mathsf{PHFE}}}, \mathsf{PP})$*: The setup algorithm is a randomized algorithm that takes as input a degree* $d \in \mathbb{N}$*, length parameter* n_{PHFE}*, and the public parameter* $\mathsf{PP} = (\mathsf{crs}, p)$*. These parameters define the function class for* PHFE*,* $\mathcal{F}_{\mathsf{PHFE},d,p,n_{\mathsf{PHFE}}}$*. It outputs a public key* PK *and a master secret key* MSK.

- $\mathsf{Enc}(\mathsf{PK}, (\mathsf{P}, \mathsf{S}) \in \mathbb{Z}_p^{n_{\mathsf{PHFE}}} \times \mathbb{Z}_p^{n_{\mathsf{PHFE}}})$*: The encryption algorithm is a randomized algorithm that takes in the public key* PK *and a message* (P, S) *and returns the ciphertext* CT. P *is considered as the public input and* S *as the secret input.* CT *is implicitly assumed to have* P *in the clear.*

- $\mathsf{KeyGen}(\mathsf{MSK}, f \in \mathcal{F}_{\mathsf{PHFE},d,p,n_{\mathsf{PHFE}}})$*: The key generation algorithm is a randomized algorithms that takes as input a degree* $(d, 2)$*-polynomial* $f \in \mathcal{F}_{\mathsf{PHFE},d,p,n_{\mathsf{PHFE}}}$ *over* $\mathbb{Z}_p$ *and returns* SK_f*, a decryption key for* f.

- $\mathsf{Dec}(\mathsf{SK}_f, \mathsf{CT})$*: The decryption algorithm is a deterministic algorithm that returns a value* out*, which is either* $\perp$ *or an integer.*

Definition 2.4.2. *(Correctness.) A PHFE scheme* PHFE *for the functionality* $\mathcal{F}_{\mathsf{PHFE}}$ *is correct if for any* $d \in \mathbb{N}$*, and polynomial* n_{PHFE}*, any security parameter* $\lambda \in \mathbb{N}$*, any* $(\mathsf{crs}, p) \leftarrow \mathsf{PPGen}(1^\lambda)$*, any* $(\mathsf{P}, \mathsf{S}) \in \mathbb{Z}_p^{n_{\mathsf{PHFE}}} \times \mathbb{Z}_p^{n_{\mathsf{PHFE}}}$*, and every function* $f \in \mathcal{F}_{\mathsf{PHFE},d,p,n_{\mathsf{PHFE}}}$ *such that* $f(\mathsf{P}, \mathsf{S}) \in \{0, 1\}$ *we have:*

$$\Pr\begin{bmatrix} (\mathsf{PK}, \mathsf{MSK}) \leftarrow \mathsf{Setup}(d, 1^{n_{\mathsf{PHFE}}}, \mathsf{PP}) \\ \mathsf{CT} \leftarrow \mathsf{Enc}(\mathsf{PK}, (\mathsf{P}, \mathsf{S})) \\ \mathsf{SK}_f \leftarrow \mathsf{KeyGen}(\mathsf{SK}, f) \\ \mathsf{Dec}(\mathsf{SK}_f, \mathsf{CT})) = f(\mathsf{P}, \mathsf{S}) \end{bmatrix} = 1.$$

Observe that the correctness of decryption is only guaranteed to hold if the value $f(\mathsf{P}, \mathsf{S})$ is in $\{0, 1\}$.

Definition 2.4.3 (Linear Efficiency)**.** *We say that* PHFE *satisfies linear efficiency if the following holds. Let* $d > 0$ *be any constant integer. Then, there exists a polynomial* poly *such that: For any polynomial* $n_{\mathsf{PHFE}}(\cdot)$ *and any parameter* $\lambda \in \mathbb{N}$*, for any* $\mathsf{PPGen}(1^\lambda) \to (\mathsf{crs}, \mathsf{PP})$ *and* $\mathsf{Setup}(d, 1^{n_{\mathsf{PHFE}}(\lambda)}, \mathsf{PP}) \to (\mathsf{PK}, \mathsf{MSK})$*, the size of the circuit* $\mathsf{Enc}(\mathsf{PK}, (\cdot, \cdot))$ *is* $O(n_{\mathsf{PHFE}}(\lambda) \cdot \mathsf{poly}(\lambda))$*.*

We now discuss the security requirement. Very roughly we want that the ciphertext should reveal only the public input P along with the outputs $f_j(\mathsf{P}, \mathsf{S})$ for every queried function f_j. This is accomplished by requiring that an encryption of (P, S) and keys for functions $\{f_j\}_{j \in Q_{\mathsf{SK}}}$ can be simulated knowing only P along with $f_j(\mathsf{P}, \mathsf{S})$ for all $j \in Q_{\mathsf{SK}}$.

Definition 2.4.4 (Simulation security)**.** *A public-key partially hiding functional encryption scheme* PHFE *for functionality* $\mathcal{F}_{\mathsf{PHFE}}$ *is (selective) SIM secure, if for every constant integer* $d > 0$ *and every polynomial* $n_{\mathsf{PHFE}} : \mathbb{N} \to \mathbb{N}$ *and* $Q_{\mathsf{SK}} : \mathbb{N} \to \mathbb{N}$*, with probability* $1 - \mathsf{negl}(\lambda)$ *over the choice of* $\mathsf{PPGen}(1^\lambda) \to \mathsf{PP} = (\mathsf{crs}, p)$*, any message* $(\mathsf{P}, \mathsf{S}) \in \mathbb{Z}_p^{n_{\mathsf{PHFE}}(\lambda)} \times \mathbb{Z}_p^{n_{\mathsf{PHFE}}(\lambda)}$ *and any choices of functions* $\{f_j\}_{j \in Q_{\mathsf{SK}}}$ *in* $\mathcal{F}_{\mathsf{PHFE},d,p,n_{\mathsf{PHFE}}}$*, the following distributions are computationally indistinguishable by any ppt algorithm with an advantage bounded by* negl_2 *for some negligible.*

$$\left\{ \left(\mathsf{PP},\ \mathsf{PK},\ \mathsf{CT},\ \{\mathsf{SK}_j\}_{j \in [Q_{\mathsf{SK}}]}\right) \;\middle|\; \begin{array}{l} (\mathsf{PK}, \mathsf{MSK}) \leftarrow \mathsf{Setup}(d, 1^{n_{\mathsf{PHFE}}}, \mathsf{PP}) \\ \mathsf{CT} \leftarrow \mathsf{Enc}(\mathsf{PK}, (\mathsf{P}, \mathsf{S})) \\ \forall j \in [Q_{\mathsf{SK}}],\ \mathsf{SK}_j \leftarrow \mathsf{KeyGen}(\mathsf{MSK}, f_j) \end{array} \right\}$$

$$\left\{ \left(\mathsf{PP},\ \widetilde{\mathsf{PK}},\ \widetilde{\mathsf{CT}},\ \{\widetilde{\mathsf{SK}_j}\}_{j \in [Q_{\mathsf{SK}}]}\right) \;\middle|\; \begin{array}{l} (\widetilde{\mathsf{PK}}, \widetilde{\mathsf{MSK}}) \leftarrow \widetilde{\mathsf{Setup}}(d, 1^{n_{\mathsf{PHFE}}}, \mathsf{PP}) \\ \widetilde{\mathsf{CT}} \leftarrow \widetilde{\mathsf{Enc}}(\widetilde{\mathsf{MSK}}, \mathsf{P}) \\ \forall j \in [Q_{\mathsf{SK}}],\ \widetilde{\mathsf{SK}_j} \leftarrow \widetilde{\mathsf{KeyGen}}(\widetilde{\mathsf{MSK}}, f_j, f_j(\mathsf{P}, \mathsf{S})) \end{array} \right\}$$

Where $\widetilde{\mathsf{Setup}}$*,* $\widetilde{\mathsf{Enc}}$*,* $\widetilde{\mathsf{KeyGen}}$ *are additional polynomial time algorithms provided by the scheme. Further the scheme is said to be subexponentially SIM secure if* negl_1 *and* negl_2 *are* $O(\exp(-\lambda^{\Omega(1)}))$*.*

2.5 Ingredient 2: Preproceseed Randomized Encoding

In this section, we define a Preprocessed Randomized Encoding scheme. We define and build it for the following function class:

Function Class: The function class $\mathcal{F}_{\mathsf{PRE}} = \{\mathcal{F}_{\mathsf{PRE},n_{\mathsf{PRE}},m_{\mathsf{PRE}},k_{\mathsf{PRE}},\lambda}\}_{n_{\mathsf{PRE}},m_{\mathsf{PRE}},k_{\mathsf{PRE}} \in \mathsf{Poly}, \lambda \in \mathbb{N}}$ is indexed with three polynomials $n_{\mathsf{PRE}}, m_{\mathsf{PRE}}, k_{\mathsf{PRE}} : \mathbb{N} \to \mathbb{N}$ and a parameter $\lambda \in \mathbb{N}$. We define this function class to be exactly $\mathcal{F}_{\mathsf{FE},n_{\mathsf{PRE}},m_{\mathsf{PRE}} \cdot k_{\mathsf{PRE}},\lambda}$, consisting of all circuits with $n_{\mathsf{PRE}}(\lambda)$ input bits and $m_{\mathsf{PRE}}(\lambda) \cdot k_{\mathsf{PRE}}(\lambda)$ output bits where every output bit is computed by a circuit of size λ.

Definition 2.5.1 (Syntax of Preprocessed Randomized Encoding)**.** *A preprocessed randomized encoding scheme* PRE *for the function class* $\mathcal{F}_{\mathsf{PRE}}$ *contains the following polynomial time algorithms:*

- $\mathsf{PRE.PreProc}(1^\lambda, 1^{n_{\mathsf{PRE}}}, 1^{m_{\mathsf{PRE}}}, 1^{k_{\mathsf{PRE}}}, p, \mathbf{x} \in \{0,1\}^{n_{\mathsf{PRE}}}) \to (\mathsf{PI}, \mathsf{SI})$. *The preprocessing algorithm takes as inputs the security parameter* λ*, input length* $1^{n_{\mathsf{PRE}}}$*, output block length* $1^{m_{\mathsf{PRE}}}$*, number of output blocks parameter* $1^{k_{\mathsf{PRE}}}$ *a prime* p *and an input* $\mathbf{x} \in \{0,1\}^n$*. It outputs preprocessed input* $(\mathsf{PI}, \mathsf{SI}) \in \mathbb{Z}_p^{\ell_{\mathsf{PRE}}}$*, where* PI *is the public part and* SI *is the private part of the input.*

- $\mathsf{PRE.Encode}(C, (\mathsf{PI}, \mathsf{SI})) = \mathbf{y}$. *The encoding algorithm takes inputs a circuit* $C \in \mathcal{F}_{\mathsf{PRE},n_{\mathsf{PRE}},m_{\mathsf{PRE}},k_{\mathsf{PRE}},\lambda}$*, and preprocessed input* $(\mathsf{PI}, \mathsf{SI})$*. It outputs a binary encoding* $\mathbf{y}$*.*

- $\mathsf{PRE.Decode}(\mathbf{y}) = \mathsf{out}$. *The decoding algorithm takes as input an encoding* $\mathbf{y}$ *and outputs a binary output* out*.*

Remark 2.5.1. Note that we could have defined the primitive without a parameter k_{PRE} by considering circuits with output length m_{PRE} as described in the high-level overview earlier. This is only done because this notation will align well with rest of the primitives that we use and build in this thesis. Instead of requiring the size of the circuit computing the preprocessing to be proportional to ${m_{\mathsf{PRE}}}^{1-\epsilon}$ for some constant $\epsilon > 0$, we will require it to be proportional to $m_{\mathsf{PRE}} \cdot {k_{\mathsf{PRE}}}^{1-\epsilon}$. By setting k_{PRE} to be sufficiently large function of m_{PRE}, this will ensure the size of the circuit computing the preprocessing is sublinear in $m_{\mathsf{PRE}} \cdot k_{\mathsf{PRE}}$

In this thesis, we care about constructions where for the function class above, $n_{\mathsf{PRE}}, m_{\mathsf{PRE}}$ and k_{PRE} are all polynomially related with λ, that is, of maginitude $\lambda^{\Theta(1)}$. Further, the output block length is super-linear in the input length, that is, $m_{\mathsf{PRE}} = {n_{\mathsf{PRE}}}^{1+\epsilon}$ for some constant $\epsilon > 0$.

Correctness and Security Requirements

Definition 2.5.2 (Correctness). *We say that* PRE *is correct if the following holds: For every* $\lambda \in \mathbb{N}$, $n_{\mathsf{PRE}}, m_{\mathsf{PRE}}, k_{\mathsf{PRE}} = \Theta(\lambda^{\Theta(1)})$, *$p$ a prime,* $\mathbf{x} \in \{0,1\}^{n_{\mathsf{PRE}}}$, *and* $C \in \mathcal{F}_{\mathsf{PRE}, n_{\mathsf{PRE}}, m_{\mathsf{PRE}}, k_{\mathsf{PRE}}, \lambda}$.

$$\Pr[\mathsf{Decode}(\mathsf{Encode}(C, \mathsf{PreProc}(1^\lambda, 1^{n_{\mathsf{PRE}}}, 1^{m_{\mathsf{PRE}}}, 1^{k_{\mathsf{PRE}}}, p, \mathbf{x}))) = C(\mathbf{x})] \geq 1 - \exp(-\lambda^{\Omega(1)}).$$

Definition 2.5.3 (Indistinguishability Security). *We say that* PRE *scheme is secure if the following holds: Let* $\beta, c_1, c_2, c_3 > 0$ *be arbitrary constants, and* $p : \mathbb{N} \to \mathbb{N}$ *be any function that takes as input any integer* r *and outputs a* r^β *bit prime and* $n_{\mathsf{PRE}}(r) = r^{c_1}$, $m_{\mathsf{PRE}}(r) = r^{c_2}$ *and* $k_{\mathsf{PRE}} = r^{c_3}$ *be three polynomials. Let* $\{C, \mathbf{x}_0, \mathbf{x}_1\}_{\lambda \in \mathbb{N}}$ *be any ensemble where* $\mathbf{x}_0, \mathbf{x}_1 \in \{0,1\}^{n_{\mathsf{PRE}}(\lambda)}$ *and* $C \in \mathcal{F}_{\mathsf{PRE}, n_{\mathsf{PRE}}(\lambda), m_{\mathsf{PRE}}(\lambda), k_{\mathsf{PRE}}(\lambda), \lambda}$ *with* $\mathbf{y} = C(\mathbf{x}_0) = C(\mathbf{x}_1)$. *Then it holds that for any* $\lambda \in \mathbb{N}$, *and letting* $p = p(\lambda)$, $n_{\mathsf{PRE}} = n_{\mathsf{PRE}}(\lambda)$, $m_{\mathsf{PRE}} = m_{\mathsf{PRE}}(\lambda)$ *and* $k_{\mathsf{PRE}} = k_{\mathsf{PRE}}(\lambda)$ *it holds that the following distributions are computationally indistinguishable*

$$\left\{(\mathsf{PI}, \mathbf{y}) \mid (\mathsf{PI}, \mathsf{SI}) \leftarrow \mathsf{PRE.PreProc}(1^\lambda, 1^{n_{\mathsf{PRE}}}, 1^{m_{\mathsf{PRE}}}, 1^{k_{\mathsf{PRE}}}, p, \mathbf{x}_0),\ \mathbf{y} \leftarrow \mathsf{PRE.Encode}(C, \mathsf{PI}, \mathsf{SI})\right\}$$
$$\left\{(\mathsf{PI}, \mathbf{y}) \mid (\mathsf{PI}, \mathsf{SI}) \leftarrow \mathsf{PRE.PreProc}(1^\lambda, 1^{n_{\mathsf{PRE}}}, 1^{m_{\mathsf{PRE}}}, 1^{k_{\mathsf{PRE}}}, p, \mathbf{x}_1),\ \mathbf{y} \leftarrow \mathsf{PRE.Encode}(C, \mathsf{PI}, \mathsf{SI})\right\}$$

Further, we say that PRE *is subexponentially secure the above distributions are subexponentially indistinguishable.*

The Efficiency and Complexity Requirements

Definition 2.5.4 (Sublinear Efficiency of PRE)**.** *We require that there exists a polynomial* poly *and constants* $c_1, c_2, c_3 > 0$ *such that for every polynomials* $n_{\mathsf{PRE}}, m_{\mathsf{PRE}}$ *and* k_{PRE} *and every security parameter* $\lambda \in \mathbb{N}$*, every prime* p*, the (randomized) circuit* $D(\cdot)$ *that on input* $\mathbf{x} \in \{0,1\}^{n_{\mathsf{PRE}}}$ *computes* $\mathsf{PRE.PreProc}(1^\lambda, 1^{n_{\mathsf{PRE}}}, 1^{m_{\mathsf{PRE}}}, 1^{k_{\mathsf{PRE}}}, p, \mathbf{x})$ *has size bounded by* $((n_{\mathsf{PRE}} + {m_{\mathsf{PRE}}}^{1-c_1}){k_{\mathsf{PRE}}}^{c_2} + m_{\mathsf{PRE}}{k_{\mathsf{PRE}}}^{1-c_3})\,\mathsf{poly}(\lambda, \log p)$*.*

In particular, this implies that when $m_{\mathsf{PRE}} = m_{\mathsf{PRE}}(\lambda) = \Theta(\lambda^{\Theta(1)})$*,* $n_{\mathsf{PRE}} = O({m_{\mathsf{PRE}}}^{1-\epsilon})$ *for some constant* $\epsilon \in (0,1)$*, then, there exists some constant* $c > 0, \gamma(c_1, c_2, c_3, c) > 0$ *such that when* $k_{\mathsf{PRE}} = {n_{\mathsf{PRE}}}^c$*, then the size of* D *is bounded by* $(m_{\mathsf{PRE}} \cdot k_{\mathsf{PRE}})^{1-\gamma} \cdot \mathsf{poly}(\lambda, \log p))$*.*

Definition 2.5.5 (Complexity of Encoding)**.** *We require that for every polynomials* $n_{\mathsf{PRE}}, m_{\mathsf{PRE}}, k_{\mathsf{PRE}}$*, every security parameter* $\lambda \in \mathbb{N}$*, every* $C \in \mathcal{F}_{\mathsf{PRE}, n_{\mathsf{PRE}}, m_{\mathsf{PRE}}, k_{\mathsf{PRE}}, \lambda}$*, and every prime* p*, there exists a polynomial mapping* f *satisfying the following:*

- *For every input* $\mathbf{x} \in \{0,1\}^{n_{\mathsf{PRE}}}$*, and every* $(\mathsf{PI}, \mathsf{SI}) \leftarrow \mathsf{PreProc}(1^\lambda, 1^{n_{\mathsf{PRE}}}, 1^{m_{\mathsf{PRE}}}, 1^{k_{\mathsf{PRE}}}, p, \mathbf{x})$*,*

$$f(\mathsf{PI}, \mathsf{SI}) \bmod p = \mathsf{PRE.Encode}(C, (\mathsf{PI}, \mathsf{SI})) \ .$$

- *There is a universal constant* $d \in \mathbb{N}$ *independent of all parameters, s.t.,* f *has degree* d *in* PI *and degree 2 in* SI*.*

- f *can be uniformly and efficiently generated from* $\lambda, n_{\mathsf{PRE}}, m_{\mathsf{PRE}}, k_{\mathsf{PRE}}, p, C$*.*

2.6 Bootstrapping to Functional Encryption

In this section, we show how construct a public-key IND-secure sublinear functional encryption scheme FE for the function class $\mathcal{F}_{\mathsf{FE}, n_{\mathsf{FE}}, m_{\mathsf{FE}}, \lambda}$ where $m_{\mathsf{FE}} = n_{\mathsf{FE}}^{1+\epsilon}$ for some constant $\epsilon > 0$ (described later) and $n_{\mathsf{FE}} = \lambda^{\Omega(1)}$ is an arbitrary polynomial in the security parameter. This class consists of all circuits with n_{FE} input bits, m_{FE} output bits, where each output bit is computed by a circuit of size λ.

Ingredients: We make use of two ingredients:

1. A PRE scheme. Let $d > 0$ be the constant degree associated with the scheme. Let $\epsilon' > 0$ be an arbitrary constant. We set:
   - $n_{\mathsf{PRE}} = n_{\mathsf{FE}}$,
   - $m_{\mathsf{PRE}} = {n_{\mathsf{PRE}}}^{1+\epsilon'}$,
   - $k_{\mathsf{PRE}} = {n_{\mathsf{PRE}}}^{c} = n_{\mathsf{FE}}^{c}$ where $c, \gamma > 0$ are constants such that the size of the encoding circuit is bounded by $(m_{\mathsf{PRE}} k_{\mathsf{PRE}})^{1-\gamma}\,\mathsf{poly}(\lambda, \log_2 p)$. Set $m_{\mathsf{FE}} = m_{\mathsf{PRE}} \cdot k_{\mathsf{PRE}} = n_{\mathsf{FE}}^{1+\epsilon'+c}$. Thus, $\epsilon = \epsilon' + c$,
2. A PHFE scheme:
   - That supports degree $(d, 2)$-polynomials,
   - Set $n_{\mathsf{PHFE}} = \ell_{\mathsf{PRE}}$ where ℓ_{PRE} is the length of PRE encoding. Observe that due to sublinear efficiency of PRE, $\ell_{\mathsf{PRE}} = O((m_{\mathsf{PRE}} \cdot k_{\mathsf{PRE}})^{1-\gamma}\,\mathsf{poly}(\lambda, \log_2 p))$.

We now describe our construction in Figure 2.1:

We now argue various properties associated with the scheme.

Parameters. Observe how the parameters for PHFE and PRE schemes are chosen. The prime p is sampled by $\mathsf{PHFE.PPGen}$ on input the security parameter 1^λ. It is a $\mathsf{poly}(\lambda)$ bit prime modulus. We set PRE parameters so that it can evaluate circuits in $\mathcal{F}_{\mathsf{FE}, n_{\mathsf{FE}}, m_{\mathsf{FE}}, \lambda}$ where $m_{\mathsf{FE}} = n_{\mathsf{FE}}^{1+\epsilon}$ while ensuring that the circuit running the preprocessing algorithm is sublinear in m_{FE}. This means that $n_{\mathsf{PRE}} = n_{\mathsf{FE}}$ and $m_{\mathsf{FE}} = m_{\mathsf{PRE}} \cdot k_{\mathsf{PRE}}$. The degree of the PHFE scheme is set to be d where $\mathsf{PRE.Encode}$ is computable by degree $(d, 2)$-polynomials. The parameter n_{PHFE} is set to be equal to ℓ_{PRE}, which is the length of the preprocessing computed by PRE.

The FE scheme

Parameter Generation $\mathsf{FE.Setup}(1^\lambda, 1^{n_{\mathsf{FE}}}, 1^{m_{\mathsf{FE}}})$: Run the following steps:

- $\mathsf{PHFE.PPGen}(1^\lambda) \to \mathsf{PHFE.PP} = (\mathsf{crs}, p)$,
- Run $\mathsf{PHFE.Setup}(d, 1^{n_{\mathsf{PHFE}}}, \mathsf{PP}) \to (\mathsf{PHFE.PK}, \mathsf{PHFE.MSK})$,
- Output $\mathsf{PK} = (\mathsf{PHFE.PK}, \mathsf{crs}, p)$ and $\mathsf{MSK} = \mathsf{PHFE.MSK}$.

Encrypt $\mathsf{FE.Enc}(\mathsf{PK}, \mathbf{x} \in \{0,1\}^{n_{\mathsf{FE}}})$: Run the following steps:

- Parse $\mathsf{PK} = (\mathsf{PHFE.PK}, \mathsf{crs}, p)$.
- Preprocess $\mathbf{x}$ using the PRE scheme, $\mathsf{PRE.PreProc}(1^\lambda, 1^{n_{\mathsf{PRE}}}, 1^{m_{\mathsf{PRE}}}, 1^{k_{\mathsf{PRE}}}, p, \mathbf{x}) \to (\mathsf{PI}, \mathsf{SI})$.
- Encrypt $\mathsf{PHFE.Enc}(\mathsf{PHFE.PK}, (\mathsf{PI}, \mathsf{SI})) \to \mathsf{CT}$. Output CT.

Keygen $\mathsf{FE.KeyGen}(\mathsf{MSK}, C)$**:** Run the following steps:

- Let $f_1, \ldots, f_T$ be degree $(d, 2)$ polynomials that compute $\mathsf{PRE.Encode}(C, (\cdot, \cdot))$.
- Compute $\mathsf{PHFE.KeyGen}(\mathsf{PHFE.MSK}, f_i) \to \mathsf{SK}_i$ for $i \in [T]$.
- Output $\mathsf{SK}_C = (\mathsf{SK}_1, \ldots, \mathsf{SK}_T)$.

Decrypt $\mathsf{FE.Dec}(\mathsf{SK}_C, \mathsf{CT})$: Run the following steps:

- Parse $\mathsf{SK}_C = (\mathsf{SK}_1, \ldots, \mathsf{SK}_T)$. For every $i \in [T]$, compute $\mathsf{PHFE.Dec}(\mathsf{SK}_i, \mathsf{CT}) = y_i$.
- Let $\mathbf{y} = (y_1, \ldots, y_T)$ and output $\mathsf{PRE.Dec}(\mathbf{y})$.

Figure 2.1: Description of the FE scheme

Correctness. Correctness follows from the correctness of PHFE and PRE scheme. The encryption algorithm encrypting $\mathbf{x}$ produces $\mathsf{PHFE.Enc}(\mathsf{PHFE.PK}, (\mathsf{PI}, \mathsf{SI}))$ where $(\mathsf{PI}, \mathsf{SI})$ is a preprocessing of input $\mathbf{x}$ using the PRE scheme. The key for a circuit C, SK_C consists of $\{\mathsf{SK}_i\}_{i\in[T]}$, where each SK_i is a PHFE key for the degree $(d, 2)$-polynomial f_i that computes the i^{th} bit y_i of $\mathsf{Encode}(C, (\mathsf{PI}, \mathsf{SI}))$. Therefore, during the decryption one produces $\mathbf{y} = \mathsf{Encode}(C, (\mathsf{PI}, \mathsf{SI}))$. Finally, the decryption outputs $\mathsf{PRE.Decode}(\mathbf{y})$, which is equal to $C(\mathbf{x})$ if the PRE scheme is correct.

Sublinearity. We now bound the size of the circuit computing the encryption of an input $\mathbf{x} \in \{0, 1\}^{n_{\mathsf{FE}}}$. The size of the circuit is:

$$\mathsf{size}_{\mathsf{PRE}} + \mathsf{size}_{\mathsf{PHFE}},$$

where $\mathsf{size}_{\mathsf{PRE}}$ is the size of the circuit computing $(\mathsf{PI}, \mathsf{SI})$, and $\mathsf{size}_{\mathsf{PHFE}}$ is the size of the circuit encrypting $(\mathsf{PI}, \mathsf{SI})$. Observe that due to sublinear efficiency of PRE:

$$\begin{aligned}\mathsf{size}_{\mathsf{PRE}} &\leq (m_{\mathsf{PRE}} \cdot k_{\mathsf{PRE}})^{1-\gamma}\, \mathsf{poly}_1(\lambda, \log_2 p)\\ &= O((m_{\mathsf{PRE}} \cdot k_{\mathsf{PRE}})^{1-\gamma}\, \mathsf{poly}_2(\lambda))\end{aligned}$$

for some other polynomial poly_2 since the bit length of p is a polynomial in λ. Also observe that due to linear efficiency of PHFE:

$$\begin{aligned}\mathsf{size}_{\mathsf{PHFE}} &\leq \ell_{\mathsf{PRE}} \cdot \mathsf{poly}_3(\lambda, \log_2 p)\\ &= O(\ell_{\mathsf{PRE}} \cdot \mathsf{poly}_4(\lambda))\end{aligned}$$

for some other polynomial poly_4 since the bit length of p is a polynomial in λ. Since $\ell_{\mathsf{PRE}} = O(\mathsf{size}_{\mathsf{PRE}})$ it holds that:

$$\mathsf{size}_{\mathsf{PRE}} + \mathsf{size}_{\mathsf{PHFE}} = O((m_{\mathsf{PRE}} \cdot k_{\mathsf{PRE}})^{1-\gamma} \cdot \mathsf{poly}_5(\lambda)).$$

Finally, since $m_{\mathsf{FE}} = m_{\mathsf{PRE}} \cdot k_{\mathsf{PRE}}$, we have:

$$\mathsf{size}_{\mathsf{PRE}} + \mathsf{size}_{\mathsf{PHFE}} = O(m_{\mathsf{FE}}^{1-\gamma} \cdot \mathsf{poly}_5(\lambda)).$$

This concludes the proof.

Security. We now prove security. Infuitively the security holds due to the security of the underlying PHFE scheme and the security of the PRE scheme. The simulation security of the PHFE ensures that the adversary only learns the encoding $\mathbf{y}$ along with the public input PI. Finally, the security of the PRE scheme ensures that $(\mathsf{PI},\ \mathbf{y})$ is indistinguishable in the case when $\mathbf{x}_0$ is encrypted versus the case when $\mathbf{x}_1$ is encrypted where $C(\mathbf{x}_0) = C(\mathbf{x}_1)$. To prove this formally, we list three hybrids, where the first hybrid is an encryption of $\mathbf{x}_b$ for a random bit $b \leftarrow \{0,1\}$ and the final hybrid is independent of b. Then, we argue indistinguishability between them. If PRE and PHFE are both subexponentially secure then so is the constructed FE.

$\mathbf{Hybrid}_0$: Let C be the circuit query and $\mathbf{x}_0, \mathbf{x}_1 \in \{0,1\}^{n_{\mathsf{FE}}}$ be the two challenge messages such that $C(\mathbf{x}_0) = C(\mathbf{x}_1)$. Generate $(\mathsf{PI}, \mathsf{SI})$ using $\mathsf{PRE.PreProc}$ algorithm, while preprocessing $\mathbf{x}_b$ for a randomly chosen bit $b \leftarrow \{0,1\}$. Generate $\mathsf{CT} = \mathsf{PHFE.Enc}(\mathsf{PHFE.PK}, (\mathsf{PI}, \mathsf{SI}))$. For the keys, compute $\{\mathsf{SK}_i \leftarrow \mathsf{PHFE.KeyGen}(\mathsf{PHFE.MSK}, f_i)\}_{i\in[T]}$. Give to the adversary $(\mathsf{PK}, \mathsf{SK}_C = (\mathsf{SK}_1, \ldots, \mathsf{SK}_T), \mathsf{CT})$.

$\mathbf{Hybrid}_1$: In this hybrid, invoke the simulator of the PHFE scheme. Simulate the public key, the secret keys and the ciphertext. Note that this can be done by knowing PI (generated as in the previous hybrid) along with $\mathbf{y} = \mathsf{PRE.Encode}(C, (\mathsf{PI}, \mathsf{SI}))$.
Observe that $\mathbf{Hybrid}_0$ is indistinguishable to $\mathbf{Hybrid}_1$ due to the security of the PHFE scheme. The only difference between the two hybrids is how $(\mathsf{PHFE.PK}, \mathsf{CT}, \mathsf{SK}_1, \ldots, \mathsf{SK}_T)$ is generated. In $\mathbf{Hybrid}_0$ they are generated using the honest algorithms, where as in $\mathbf{Hybrid}_1$, they are simulated using $\big\{\mathsf{PI}, \{f_i, y_i = f_i(\mathsf{PI}, \mathsf{SI})\}_{i\in[T]}\big\}$.

$\mathbf{Hybrid}_2$: In this hybrid, generate $(\mathsf{PI}, \mathbf{y})$ by first computing $(\mathsf{PI}, \mathsf{SI})$ to preprocess $\mathbf{x}_0$, and then computing $\mathbf{y} = \mathsf{PRE.Encode}(C, (\mathsf{PI}, \mathsf{SI}))$. This hybrid is independent of b. $\mathbf{Hybrid}_1$ is indistinguishable to $\mathbf{Hybrid}_2$ due to the security of the PRE scheme. The only difference between these hybrids is how $(\mathsf{PI}, \mathbf{y})$ are generated. In $\mathbf{Hybrid}_1$, they are generated by using

$\mathbf{x}_b$, where as in $\mathbf{Hybrid}_2$ they are generated using $\mathbf{x}_0$. Note that $C(\mathbf{x}_b) = C(\mathbf{x}_0)$, and thus the indistinguishability follows from the security of the PRE scheme.

This proves the following result:

Lemma 2.6.1. *Assuming the existence of a* PHFE *scheme as in Definition 2.4.1 and a* PRE *scheme as in Definition 2.5.1, there exists a sublinear FE scheme for* $\mathcal{F}_{\mathsf{FE},\lambda,n_{\mathsf{FE}},n_{\mathsf{FE}}^{1+\epsilon}}$ *for some constant* $\epsilon > 0$*. If the underlying primitives are subexponentially secure then so is the resulting FE scheme.*

Using Thereom 2.3.3 and the lemma above we have the following Lemma:

Lemma 2.6.2. *Assuming the existence of a* PHFE *scheme as in Definition 2.4.1 and a* PRE *scheme as in Definition 2.5.1, there exists a sublinear FE scheme for* $\mathcal{F}_{\mathsf{FE},\lambda,n_{\mathsf{FE}},n_{\mathsf{FE}}^{1+\epsilon}}$ *for some constant* $\epsilon > 0$*. If the underlying primitives are subexponentially secure then there exists a secure indistinguishability obfuscation for all circuits.*

In Theorem 6.2.2, it is shown that PHFE can be constructed using DLIN assumption over prime order symmetric bilinear groups (Definition 6.1.2). In Theorem 3.1.2, it is shown that a PRE scheme can be constructed assuming PRG in NC^0 (Definition 5.2.1) and δ-LPN assumption for any constant $\delta > 0$ (Definition 4.1.1). As a consequence, we have the following Theorem:

Theorem 2.6.1. *If there exists constants* $\delta, \tau > 0$ *such that:*

- δ-LPN *assumption holds (Definition 4.1.1),*
- *There exists a* PRG *in* NC^0 *with a stretch of* $n^{1+\tau}$ *where* n *is length of the input (Definition 5.2.1),*
- *The* DLIN *assumption over prime order symmetric bilinear groups (Definition 6.1.2) holds.*

Then, there exists a sublinear functional encryption scheme for $\mathcal{F}_{\mathsf{FE},\lambda,n_{\mathsf{FE}},n_{\mathsf{FE}}^{1+\epsilon}}$ *for some constant* $\epsilon > 0$. *Further if the underlying assumptions are subexponentially secure, then there exists a secure indistinguishability obfuscation for all circuits.*

2.7 Outline

The outline of rest of the sections can be summarized in Figure 2.2. In Chapter 6, we show the construction of the PHFE scheme assuming the DLIN assumption. In order to build a PRE scheme, we introduce two abstractions to achieve a modular presentation with respect to where exactly our assumptions are used. Our first abstraction is a Preprocessed Polynomial Encoding scheme (PPE) and our second abstraction is an Amortized Randomized Encoding scheme (ARE). In Chapter 3, we motivate and formally define an ARE and a PPE scheme and construct our PRE scheme from these two abstractions. In Chapter 4 we construct our PPE scheme relying on the LPN assumption. Finally, in Chapter 5 we construct our ARE scheme relying on the PRG in NC^0 assumption. This concludes our overall outline.

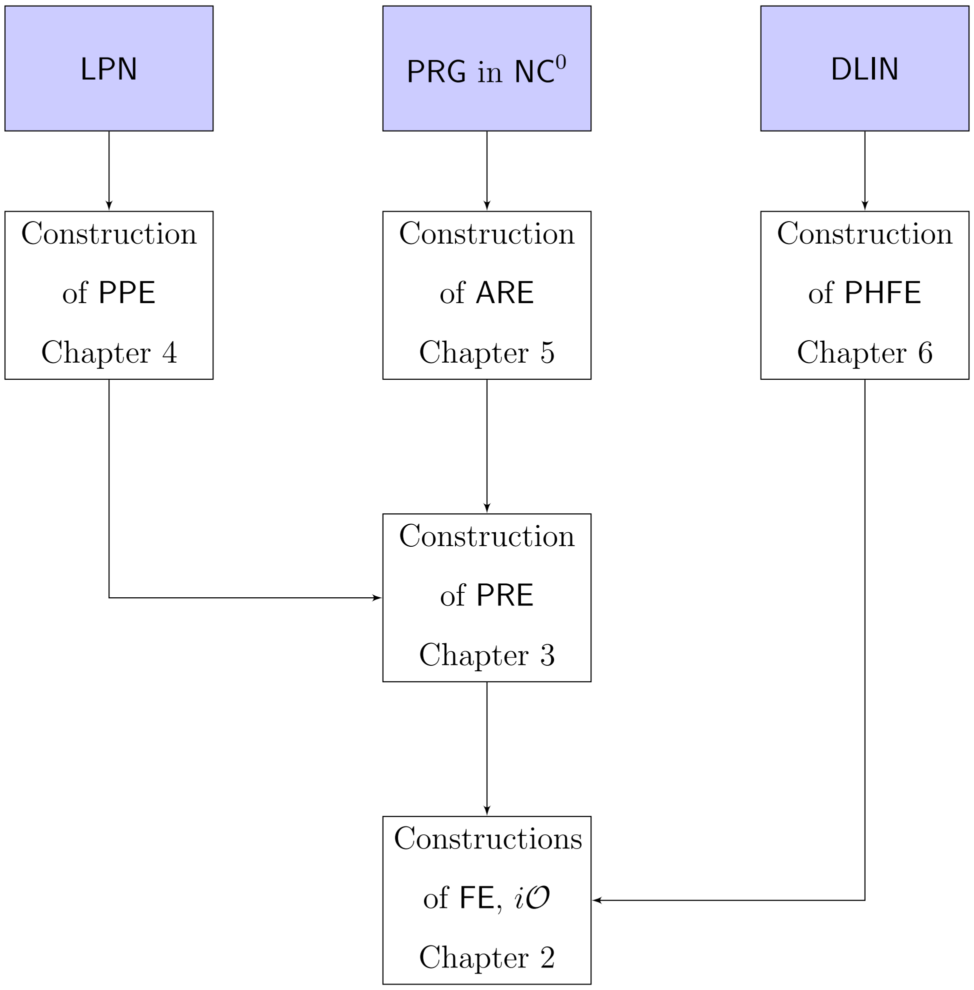
LPN
PRG in NC^0
DLIN
Construction of PPE Chapter 4
Construction of ARE Chapter 5
Construction of PHFE Chapter 6
Construction of PRE Chapter 3
Constructions of FE, $i\mathcal{O}$ Chapter 2

Figure 2.2: Flowchart depicting the technical outline.

CHAPTER 3

Preprocessed Randomized Encoding

In this chapter we discuss how to construct a Preprocessed Randomized Encoding scheme PRE. We will describe (at an intuition level) two building blocks that will suffice to build a PRE scheme and show how exactly this transformation works. In future chapters we will describe in formal details how to construct those two building blocks.

3.1 Technical Outline: Preprocessed Randomized Encoding

Recall the goal of a PRE scheme. In a PRE scheme we care about the following function class $\mathcal{F}_{\mathsf{PRE}} = \mathcal{F}_{\mathsf{PRE},n_{\mathsf{PRE}},m_{\mathsf{PRE}},k_{\mathsf{PRE}},\lambda}$ which consists of all circuits C with n_{PRE} inputs, $m_{\mathsf{PRE}} \cdot k_{\mathsf{PRE}}$ output bits where the size of circuit computing each output bit is λ. It satisfies the following syntactical and structural requirements:

1. $\mathsf{PRE.PreProc}(\mathbf{x}, p)$: The preprocessing algorithm takes as input $\mathbf{x} \in \{0,1\}^{n_{\mathsf{PRE}}}$ and it outputs two vectors $(\mathsf{PI}, \mathsf{SI})$ over $\mathbb{Z}_p^{\ell_{\mathsf{PRE}}}$. The size of the circuit computing this should be $\tilde{O}((n_{\mathsf{PRE}} + {m_{\mathsf{PRE}}}^{1-c_1}){k_{\mathsf{PRE}}}^{c_2} + m_{\mathsf{PRE}}{k_{\mathsf{PRE}}}^{1-c_3})$ where $\tilde{O}$ hides polynomial factors in λ and $\log_2 p$ and $c_1, c_2, c_3 > 0$ are constants.

2. $\mathsf{PRE.Encode}(C, (\mathsf{PI}, \mathsf{SI}))$: The encoding algorithm takes as input $(\mathsf{PI}, \mathsf{SI})$ and a circuit C in the function class $\mathcal{F}_{\mathsf{PRE}}$. It outputs a string $\mathbf{y} \in \{0,1\}^{T_{\mathsf{PRE}}}$. Further, each output bit y_i is computable by an efficiently computable degree $(d, 2)$-polynomial in $(\mathsf{PI}, \mathsf{SI})$ over $\mathbb{Z}_p$.

3. $\mathsf{PRE.Decode}(\mathbf{y})$: The decoding algorithm takes as input a string $\mathbf{y}$, and it outputs $C(\mathbf{x})$.

4. For any $\mathbf{x}_0, \mathbf{x}_1$ such that $C(\mathbf{x}_0) = C(\mathbf{x}_1)$, it holds that $(\mathsf{PI}_0, \mathbf{y}_0)$ is computationally indistinguishable to $(\mathsf{PI}_1, \mathbf{y}_1)$ where for $b \in \{0,1\}$, PI_b and $\mathbf{y}_b$ is computed by using input $\mathbf{x}_b$.

As described before, the requirements of PRE are extremely similar to a randomized encoding in NC^0 scheme. The main difference is that PRE has the following additional relaxations:

- (*Preprocessing model and a public input*): It is allowed to preprocess the input $\mathbf{x}$ into an efficiently computable preprocessing $(\mathsf{PI}, \mathsf{SI})$ which now has an additional public component PI along with the secret component SI. Here, PI should computationally hide $\mathbf{x}$, and,

- (*Complexity transferred to the public component*): For any circuit C, the degree of the polynomial computing the encoding $\mathbf{y}$ over SI is now just two, whereas its degree over PI could be an arbitrary constant.

As a result of this connection with an RE scheme we take the following approach: we devise a "secure polynomial evaluation mechanism" which we call a Preprocessed Polynomial Encoding PPE. This mechanism takes as input an RE scheme in NC^0 which is computable by degree d polynomials (and more generally, description of some wide family of constant degree d polynomials $\mathcal{F}_{\mathsf{PPE}}$). It compiles any input $\mathbf{x} \in \{0,1\}^{n_{\mathsf{PPE}}}$ and preprocesses it efficiently using a sublinear sized circuit into two vectors (P, S) over $\mathbb{Z}_p$ such that: i) P computationally hides information about $\mathbf{x}$; ii) For any $f \in \mathcal{F}_{\mathsf{PPE}}$, there exists a degree $(d,2)$-polynomial g_f such that $g_f(\mathsf{P}, \mathsf{S}) = f(\mathbf{x})$.

Thus, in order to build a PRE scheme we first build a PPE scheme for a large class of polynomials $\mathcal{F}_{\mathsf{PPE}}$ and then build a randomized encoding scheme RE where the encoding functions are computable in $\mathcal{F}_{\mathsf{PPE}}$. It turns out that an arbitrary randomized encoding in NC^0 does not work for us, but a simple adaption of Yao's garbled circuits [Yao86] does. We call such a randomized encoding scheme an amortized randomized encoding scheme. We formalize the notion of PPE along with the exact function class $\mathcal{F}_{\mathsf{PPE}}$ in Section 3.1.1 and amortized randomized encoding in Section 3.1.2. In Section 3.1.3 we describe how to use both these objects to construct a PRE scheme. To build a PPE scheme, we require the LPN assumption over $\mathbb{Z}_p$, and in order to build an amortized RE scheme, we will use PRGs in NC^0. Thus, a PRE can be built using these two assumptions. We show formally how to construct PPE in Chapter 4 and an amortized RE in Chapter 5.

3.1.1 Preprocessed Polynomial Encoding

In this section, we formally define a PPE scheme. Before we formally define the notion we introduce the function class $\mathcal{F}_{\mathsf{PPE}}$. We first define the notion of a degree d monomial pattern $\mathcal{Q}$ over n variables which is just a collection of monomials of degree at most d.

Definition 3.1.1 (d-monomial pattern and monomials)**.** *For an integer $d > 0$, and an integer $n > d \in \mathbb{N}$, we say $\mathcal{Q}$ is a d-monomial pattern over n variables, if $\mathcal{Q} = \{Q_1, \ldots, Q_m\}$, where for every $i \in [m]$, we have that $0 < |Q_i| \leq d$, and each Q_i is a distinct subset of $[n]$. For any input $\mathbf{x} \in \{0,1\}^n$ and a set $Q \subseteq [n]$, define $\mathsf{Mon}_Q(\mathbf{x}) = \prod_{i \in Q} x_i$ to be the monomial in $\mathbf{x}$ corresponding to the set Q. Thus, for any input $\mathbf{x}$, a d-monomial pattern $\mathcal{Q} = \{Q_1, \ldots, Q_m\}$ over n variables defines m monomials of degree at most d.*

We denote by $\Gamma_{d,n}$ the set of all d-monomial patterns over n variables.

Definition 3.1.2 (Polynomial Class $\mathcal{F}_{\mathsf{PPE}}$)**.** *For a constant $d \in \mathbb{N}$, the family of classes of polynomials $\mathcal{F}_{\mathsf{PPE},d} = \{\mathcal{F}_{\mathsf{PPE},d,n_{\mathsf{PPE}},\mathcal{Q},k_{\mathsf{PPE}}}\}_{d \leq n_{\mathsf{PPE}} \in \mathbb{N}, \mathcal{Q} \in \Gamma_{d,n_{\mathsf{PPE}}}, k_{\mathsf{PPE}} \in [\mathbb{N}]}$ consists of polynomials $f \in \mathcal{F}_{\mathsf{PPE},d,n_{\mathsf{PPE}},\mathcal{Q},k_{\mathsf{PPE}}}$ of the following kind: f is defined by a sequence of integers $(\zeta_i^{(j)})_{j \in [k_{\mathsf{PPE}}], i \in [m_{\mathsf{PPE}}]}$. It takes as input $\mathbf{x}$ consisting of k_{PPE} blocks $\mathbf{x} = (\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(k_{\mathsf{PPE}})})$ each of n_{PPE} variables,* *and has form:*

$$f(\mathbf{x}) := \sum_{j\in[k_{\mathsf{PPE}}],\ Q_i\in\mathcal{Q}} \zeta_i^{(j)} \mathsf{Mon}_{Q_i}(\mathbf{x}^{(j)}),$$

where $\mathcal{Q}$ is a d-monomial pattern with $|\mathcal{Q}| = m_{\mathsf{PPE}}$.

In a nutshell, $\mathcal{F}_{\mathsf{PPE}}$ consists of polynomials that take as input a k_{PPE} blocks of inputs of size n_{PPE}, and computes all polynomials that are linear combination of some fixed constant degree d monomials on those inputs governed by a set $\mathcal{Q}$. Looking ahead, for the PPE scheme we will require that the size of the circuit computing (P, S) will be sublinear in $|\mathcal{Q}| \cdot k_{\mathsf{PPE}}$.

Definition 3.1.3 (Syntax of PPE)**.** *For any constant $d > 0$, a PPE scheme for function class $\mathcal{F}_{\mathsf{PPE},d}$ consists of the following p.p.t. algorithms:*

- $(\mathsf{P}, \mathsf{S}) \leftarrow \mathsf{PreProc}(1^{n_{\mathsf{PPE}}},\ 1^{k_{\mathsf{PPE}}},\ p,\ \mathcal{Q}, \mathbf{x} \in \mathbb{Z}_p^{n_{\mathsf{PPE}}\cdot k_{\mathsf{PPE}}})$: *The randomized Pre-processing algorithm takes as input the block length parameter n_{PPE}, the number of blocks parameter k_{PPE}, a prime p, a d-monomial pattern on n_{PPE} variables $\mathcal{Q}$ of size m_{PPE}, and an input $\mathbf{x} \in \mathbb{Z}_p^{n_{\mathsf{PPE}}\cdot k_{\mathsf{PPE}}}$. It processes it to output two strings, a public string P and a private string S. Both these strings are vectors over $\mathbb{Z}_p$. We denote by $\ell_{\mathsf{PPE}} = \ell_{\mathsf{PPE}}(n_{\mathsf{PPE}}, m_{\mathsf{PPE}}, k_{\mathsf{PPE}})$ the combined dimension of (P, S) over $\mathbb{Z}_p$.*
- $y \leftarrow \mathsf{Eval}(f \in \mathcal{F}_{\mathsf{PPE},d,n_{\mathsf{PPE}},\mathcal{Q},k_{\mathsf{PPE}}}, (\mathsf{P}, \mathsf{S}))$: *The deterministic evaluation algorithm takes as input the description of a function $f \in \mathcal{F}_{d,n_{\mathsf{PPE}},\mathcal{Q},k_{\mathsf{PPE}}}$ and a pre-processed input (P, S). It outputs $y \in \mathbb{Z}_p$.*

The correctness requirement is completely straightforward namely y should be equal to $f(\mathbf{x})$ with high probability.

Definition 3.1.4 ((Statistical) Correctness of PPE)**.** *Let $d > 0$ be a constant integer, a PPE scheme for the function class $\mathcal{F}_{d,\mathsf{PPE}}$ satisfies correctness if: For every $k_{\mathsf{PPE}} \in \mathbb{N}$, $n_{\mathsf{PPE}} = k^{\Theta(1)}$, and $\mathcal{Q} \in \Gamma_{d,n_{\mathsf{PPE}}}$ with $m_{\mathsf{PPE}} \geq 1$ sets, any function $f \in \mathcal{F}_{d,\mathsf{PPE},n_{\mathsf{PPE}},\mathcal{Q},k_{\mathsf{PPE}}}$, any prime p*

and any input $\mathbf{x} \in \mathbb{Z}_p^{n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}}$*:*

$$\Pr\left[\ \mathsf{Eval}(f,\ (\mathsf{P},\mathsf{S})) = f(\mathbf{x}) \bmod p \ \middle|\ (\mathsf{P},\mathsf{S}) \leftarrow \mathsf{PreProc}(1^{n_{\mathsf{PPE}}}, 1^{k_{\mathsf{PPE}}}, p, \mathcal{Q}, \mathbf{x})]\ \right] \geq 1 - O(\exp(-{k_{\mathsf{PPE}}}^{\Omega(1)}))$$

Note that we require correctness to hold when k_{PPE} is large enough, we will also require the security to hold for large values of k_{PPE}. The next definition we discuss is that of security. The security definition roughly requires that for any input $\mathbf{x} \in \mathbb{Z}_p^{n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}}$, the public part of the computed pre-processed input while pre-processing $\mathbf{x}$ is computationally indistinguishable to the public part of the pre-processed input when the pre-procssing is done for the input $0^{n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}}$.

Definition 3.1.5 (Security of PPE). *Let $d > 0$ be an integer constant. A* PPE *scheme is secure, if the following holds: Let $\beta > 0$ be any constant and $p : \mathbb{N} \to \mathbb{N}$ be any function that on input an integer r, outputs an r^β bit prime. Let $n_{\mathsf{PPE}} = k_{\mathsf{PPE}}^{\Theta(1)}$ be any polynomial in k_{PPE}. Let $p = p(k_{\mathsf{PPE}})$ and $\{\mathbf{x}_{k_{\mathsf{PPE}}}\}_{k_{\mathsf{PPE}} \in \mathbb{N}}$ be any ensemble of inputs where each $\mathbf{x}_{k_{\mathsf{PPE}}} \in \mathbb{Z}_p^{n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}}$ and $\{\mathcal{Q}_{k_{\mathsf{PPE}}}\}_{k_{\mathsf{PPE}} \in \mathbb{N}}$ be ensemble of monomial patterns with $\mathcal{Q}_{k_{\mathsf{PPE}}} \in \Gamma_{d,n_{\mathsf{PPE}}}$ with size $m_{\mathsf{PPE}} \geq 1$. Then for $k_{\mathsf{PPE}} \in \mathbb{N}$, it holds that for any probabilistic polynomial time adversary, following distributions are computationally indistinguishable with the advantage bounded by* $\mathsf{negl}(k_{\mathsf{PPE}})$.

$$\left\{\mathsf{P} \mid (\mathsf{P},\ \mathsf{S}) \leftarrow \mathsf{PreProc}(1^{n_{\mathsf{PPE}}},\ 1^{k_{\mathsf{PPE}}},\ p,\ \mathcal{Q}_{k_{\mathsf{PPE}}},\ \mathbf{x}_{k_{\mathsf{PPE}}})\right\}_{k_{\mathsf{PPE}}}$$
$$\left\{\mathsf{P} \mid (\mathsf{P},\mathsf{S}) \leftarrow \mathsf{PreProc}(1^{n_{\mathsf{PPE}}},\ 1^{k_{\mathsf{PPE}}},\ p,\ \mathcal{Q}_{k_{\mathsf{PPE}}}, 0^{n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}})\right\}_{k_{\mathsf{PPE}}}$$

Further, the scheme is said to be subexponentially secure if $\mathsf{negl}(k_{\mathsf{PPE}}) = \exp(-k_{\mathsf{PPE}}^{\Omega(1)})$.

Definition 3.1.6 (Sublinear Pre-processing Efficiency). *Let $d > 0$ be a constant integer. We say that* PPE *scheme for $\mathcal{F}_{\mathsf{PPE},d}$ satisfies sublinear efficiency if there exists a polynomial* poly *and constants $c_1, c_2, c_3 > 0$ such that for $n_{\mathsf{PPE}}, k_{\mathsf{PPE}} \in \mathbb{N}$, $\mathcal{Q} \in \Gamma_{d,n_{\mathsf{PPE}}}$ with size $m_{\mathsf{PPE}} \geq 1$ and a prime p the size of the circuit computing* $\mathsf{PreProc}(1^{n_{\mathsf{PPE}}},\ 1^{k_{\mathsf{PPE}}},\ p,\ \mathcal{Q}, \cdot)$ *is* $t_{\mathsf{PPE}} = O((n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}^{c_1} + m_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}^{1-c_2} + k_{\mathsf{PPE}}^{c_3})\,\mathsf{poly}(\log_2 p))$.

The reason we call this requirement as sublinear pre-processing efficiency is that if $m_{\mathsf{PPE}} = n_{\mathsf{PPE}}^{1+\Omega(1)}$, then, one can find a small enough $k_{\mathsf{PPE}} = n_{\mathsf{PPE}}^{\Omega(1)}$ such that $t_{\mathsf{PPE}} = \tilde{O}((m_{\mathsf{PPE}} k_{\mathsf{PPE}})^{1-\Omega(1)})$

where $\tilde{O}$ hides polynomial factors in $\log_2 p$. Finally we present the requirement that the evaluation for any function f, can be done by a constant degree polynomial g_f that is just degree two in S.

Definition 3.1.7 (Complexity of Evaluation). *Let $d \in \mathbb{N}$ be any constant. We require that* PPE *scheme for $\mathcal{F}_{\mathsf{PPE},d}$ satisfies the following. We require that for every $k_{\mathsf{PPE}} \in \mathbb{N}$, $n_{\mathsf{PPE}} = k_{\mathsf{PPE}}^{\Theta(1)}$, and $\Gamma \in \Gamma_{d,n_{\mathsf{PPE}}}$ of size $m_{\mathsf{PPE}} \geq 1$, any prime p, any input $\mathbf{x} \in \mathbb{Z}_p^{n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}}$, any pre-processed input $(\mathsf{P}, \mathsf{S}) \leftarrow \mathsf{PreProc}(1^{n_{\mathsf{PPE}}},\ 1^{k_{\mathsf{PPE}}},\ p,\ \Gamma,\ \mathbf{x})$, and any $f \in \mathcal{F}_{d,n_{\mathsf{PPE}},\mathcal{Q},k_{\mathsf{PPE}}}$, the following relation is satisfied:*

$$\mathsf{Eval}(f,\ (\mathsf{P}, \mathsf{S})) = g_{f,\mathcal{Q}}(\mathsf{P}, \mathsf{S}) \mod p$$

where $g_{f,\mathcal{Q}}(\cdot, \cdot)$ is an efficiently computable (multivariate) polynomial over $\mathbb{Z}_p$ of degree $O(d)$ in P *and degree 2 in* S.

3.1.2 Amortized Randomized Encoding

We now formally define the notion of an amortized RE scheme (which we will denote by ARE). The notion is designed to be exactly compatible with a PPE scheme. The function class $\mathcal{F}_{\mathsf{ARE}}$ is identical to the class for the PRE scheme $\mathcal{F}_{\mathsf{PRE}}$. Namely, $\mathcal{F}_{\mathsf{ARE}} = \{\mathcal{F}_{\mathsf{ARE},n_{\mathsf{ARE}},m_{\mathsf{ARE}},k_{\mathsf{ARE}},\lambda}\}_{n_{\mathsf{ARE}},k_{\mathsf{ARE}},m_{\mathsf{ARE}},\lambda \in \mathbb{N}}$ consists of all circuits $C : \{0,1\}^{n_{\mathsf{ARE}}} \rightarrow \{0,1\}^{m_{\mathsf{ARE}} \cdot k_{\mathsf{ARE}}}$ where every bit of the output is computed by a circuit of size λ. Such an ARE scheme has the following syntax:

Definition 3.1.8 (Syntax of ARE). *An* ARE *scheme consists of the following p.p.t. algorithms:*

- $\mathsf{Encode}(C \in \mathcal{F}_{\mathsf{ARE},n_{\mathsf{ARE}},m_{\mathsf{ARE}},k_{\mathsf{ARE}},\lambda}, \mathbf{x} \in \{0,1\}^{n_{\mathsf{ARE}}}) \rightarrow \mathbf{y}$. *The encoding algorithm is a randomized algorithm that takes as input a circuit $C \in \mathcal{F}_{\mathsf{ARE},n_{\mathsf{ARE}},m_{\mathsf{ARE}},k_{\mathsf{ARE}},\lambda}$ along with an input $\mathbf{x} \in \{0,1\}^{n_{\mathsf{ARE}}}$. It outputs a string $\mathbf{y} \in \{0,1\}^*$.*

- $\mathsf{Decode}(1^\lambda, 1^{n_{\mathsf{ARE}}}, 1^{m_{\mathsf{ARE}}}, 1^{k_{\mathsf{ARE}}}, \mathbf{y}) \to \mathbf{z}$: *The deterministic decode algorithm takes as input a string* $\mathbf{y}$. *It outputs* $\mathbf{z} \in \perp \cup \{0,1\}^{m_{\mathsf{ARE}} \cdot k_{\mathsf{ARE}}}$.

An ARE scheme satisfies the following properties.

Definition 3.1.9 ((Perfect) Correctness of ARE)**.** *A* ARE *scheme for the function class* $\mathcal{F}_{\mathsf{ARE}}$ *satisfies correctness if: For every polynomials* $n_{\mathsf{ARE}}(\cdot), m_{\mathsf{ARE}}(\cdot), k_{\mathsf{ARE}}(\cdot)$, *every* $\lambda \in \mathbb{N}$, *let* $n_{\mathsf{ARE}} = n_{\mathsf{ARE}}(\lambda), m_{\mathsf{ARE}} = m_{\mathsf{ARE}}(\lambda), k_{\mathsf{ARE}} = k_{\mathsf{ARE}}(\lambda)$. *Then, for every* $\mathbf{x} \in \{0,1\}^{n_{\mathsf{ARE}}}$, $C \in \mathcal{F}_{\mathsf{ARE}, n_{\mathsf{ARE}}, m_{\mathsf{ARE}}, k_{\mathsf{ARE}}, \lambda}$:

$$\Pr\left[\mathsf{Decode}(1^\lambda, 1^{n_{\mathsf{ARE}}}, 1^{m_{\mathsf{ARE}}}, 1^{k_{\mathsf{ARE}}}, \mathbf{y}) = C(\mathbf{x}) \;\middle|\; \mathbf{y} \leftarrow \mathsf{Encode}(C, \mathbf{x}) \right] = 1$$

Definition 3.1.10 (Indistinguishability Security)**.** *We say that* ARE *scheme is secure if the following holds: Let* $\lambda \in \mathbb{N}$ *be the security parameter, and* $n_{\mathsf{ARE}}, m_{\mathsf{ARE}}, k_{\mathsf{ARE}} = \Theta(\lambda^{\Theta(1)})$ *be polynomials in* λ. *For every sequence* $\{C, \mathbf{x}_0, \mathbf{x}_1\}_\lambda$ *where* $\mathbf{x}_0, \mathbf{x}_1 \in \{0,1\}^{n_{\mathsf{ARE}}}$ *and* $C \in \mathcal{F}_{\mathsf{ARE}, n_{\mathsf{ARE}}, m_{\mathsf{ARE}}, k_{\mathsf{ARE}}, \lambda}$ *with* $C(\mathbf{x}_0) = C(\mathbf{x}_1)$, *it holds that for* $\lambda \in \mathbb{N}$ *the following distributions are computationally indistinguishable*

$$\{\mathbf{y} \mid \mathbf{y} \leftarrow \mathsf{ARE.Encode}(C, \mathbf{x}_0)\}$$
$$\{\mathbf{y} \mid \mathbf{y} \leftarrow \mathsf{ARE.Encode}(C, \mathbf{x}_1)\}$$

Further, we say that ARE *is subexponentially secure the above distributions are subexponentially indistinguishable.*

Efficiency Properties. We require that such an ARE scheme is compatible with a PPE scheme. Namely, the encoding operation $\mathsf{Encode}(C, \cdot)$ uses a constant degree d-monomial pattern $\mathcal{Q}$ of small size $m'_{\mathsf{ARE}} = O((n_{\mathsf{ARE}} + m_{\mathsf{ARE}})\,\mathsf{poly}(\lambda))$ over $n'_{\mathsf{ARE}} = O((n_{\mathsf{ARE}} + m_{\mathsf{ARE}}^{1-\Omega(1)})\,\mathsf{poly}(\lambda))$ variables such that every bit is computable using those monomials. Namely:

Definition 3.1.11 (Efficiency)**.** *We require that there exists constants* $d \in \mathbb{N}, c_1, c_2 > 0$, *such that the following holds. For any* $\lambda \in \mathbb{N}$ *and any* $n_{\mathsf{ARE}}, k_{\mathsf{ARE}}, m_{\mathsf{ARE}} = \lambda^{\Omega(1)}$, *there exists* *an efficiently samplable degree d-monomial pattern* $\mathcal{Q}$ *of size* $m'_{\mathsf{ARE}} = O((n_{\mathsf{ARE}} + m_{\mathsf{ARE}})\lambda^{c_1})$ *such that for any circuit* $C \in \mathcal{F}_{\mathsf{ARE},n_{\mathsf{ARE}},m_{\mathsf{ARE}},k_{\mathsf{ARE}},\lambda}$ *and input* $\mathbf{x} \in \{0,1\}^{n_{\mathsf{ARE}}}$, $\mathsf{Encode}(C, \mathbf{x}; \mathbf{r}) \to \mathbf{y} \in \{0,1\}^T$ *satisfies the following requirements:*

- *Parse* $\mathbf{r} = (\mathbf{r}_1, \ldots, \mathbf{r}_{k_{\mathsf{ARE}}})$ *where each component is of equal size. Let* $\mathbf{a}_i = (\mathbf{x}, \mathbf{r}_i)$*. Then the length of* $\mathbf{a}_i \in \{0,1\}^{n'_{\mathsf{ARE}}}$ *is* $n'_{\mathsf{ARE}} = O((n_{\mathsf{ARE}} + m_{\mathsf{ARE}}^{1-c_2})\lambda^{c_1})$*.*

- *For* $i \in [T]$*, each* $y_i = \sum_{Q \in \mathcal{Q}, j \in [k_{\mathsf{ARE}}]} \mu_{i,Q,j} \cdot \mathsf{Mon}_Q(\mathbf{a}_j)$ *for efficiently samplable* $\mu_{i,Q,j} \in \mathbb{Z}$*.*

The first property is to ensure that $\mathbf{a}_i$ for $i \in [k_{\mathsf{ARE}}]$ will be the k_{ARE} blocks that will be preprocessed by the PPE scheme in our construction of PRE. The monomial pattern used by the PRE will be $\mathcal{Q}$, and it will be used to compute $\mathbf{y}$.

3.1.3 Construction of Preprocessed Randomized Encoding

The construction of a PRE scheme is really straightforward. We simply compose PPE with ARE. Let's take a look at it formally. Let the function class we are interested in is $\mathcal{F}_{\mathsf{PRE},n_{\mathsf{PRE}},m_{\mathsf{PRE}},k_{\mathsf{PRE}},\lambda}$ where λ is the security parameter and $n_{\mathsf{PRE}}, m_{\mathsf{PRE}}, k_{\mathsf{PRE}}$ are polynomials in the security parameter. Let p denote the prime to be used for the PRE scheme.

Ingredients: We make use of two ingredients:

1. A ARE scheme. Let $d > 0$ be the constant degree which is the degree of evaluation of the PRE scheme. We set:

    - $n_{\mathsf{ARE}} = n_{\mathsf{PRE}}$,
    - $m_{\mathsf{ARE}} = m_{\mathsf{PRE}}$,
    - $k_{\mathsf{ARE}} = k_{\mathsf{PRE}}$,
    - $m'_{\mathsf{ARE}} = (n_{\mathsf{PRE}} + m_{\mathsf{PRE}}) \cdot \lambda^{c_1}$,

- $n'_{\mathsf{ARE}} = (n_{\mathsf{PRE}} + {m_{\mathsf{PRE}}}^{1-c_2})\lambda^{c_1}$, where $c_1, c_2 > 0$ are constants associated with the efficiency requirements of ARE. Let $\mathcal{Q}_{\mathsf{ARE}}$ be the d-monomial pattern of size m'_{ARE} over n'_{ARE} variables associated with the encoding operation.

2. A PPE scheme, where we set:

- The prime to be used as p,
- $n_{\mathsf{PPE}} = n'_{\mathsf{ARE}}$,
- $m_{\mathsf{PPE}} = m'_{\mathsf{ARE}}$,
- Set the monomial pattern $\mathcal{Q}_{\mathsf{PPE}} = \mathcal{Q}_{\mathsf{ARE}} = \mathcal{Q}$. The degree of the monomial pattern is d,
- Let $d' = O(d)$ be the constant degree of the polynomial $g_f(\cdot) = \mathsf{PPE.Eval}(f, \cdot) \mod p$ used to evaluate any polynomial $f \in \mathcal{F}_{d,\mathsf{PPE},n_{\mathsf{PPE}},\mathcal{Q},k_{\mathsf{PPE}}}$.

We now describe our construction in Figure 3.1:

We now argue various properties associated with the scheme.

Correctness: Correctness follows from the correctness of the ARE and PPE scheme. The PreProc operation simply runs PPE.PreProc to preprocess $\mathbf{a} = (\mathbf{a}_1, \ldots, \mathbf{a}_{k_{\mathsf{ARE}}})$ where $\mathbf{a}_i = (\mathbf{x}, \mathbf{r}_i)$ to compute P, S. Then, when one runs $\mathsf{PRE.Encode}(C, (\mathsf{P}, \mathsf{S}))$, with overwhelming probability the output consists of $y_i = f_i(\mathbf{a})$ for $i \in [T]$, due to correctness of PPE scheme. This is same as $\mathsf{ARE.Encode}(C, \mathbf{x}, \mathbf{r}_1, \ldots, \mathbf{r}_{k_{\mathsf{ARE}}})$ by the correctness of the ARE scheme. Finally $\mathsf{PRE.Decode}(\mathbf{y}) = \mathsf{ARE.Decode}(\mathbf{y})$ which is equal to $C(\mathbf{x})$.

We now argue security,

Security The security of the PRE scheme follows almost immediately from the security of the ARE scheme. We show this by providing four hybrids and arguing indistinguishability between them. The first hybrid corresponds to the case when $\mathbf{x}_b$ is preprocessed for a random

The PRE scheme

Preprocessing $\mathsf{PRE.PreProc}(1^\lambda, 1^{n_{\mathsf{PRE}}}, 1^{m_{\mathsf{PRE}}}, 1^{k_{\mathsf{PRE}}}, p, \mathbf{x} \in \{0,1\}^{n_{\mathsf{PRE}}})$: Run the following steps:

- Sample uniformly randomness $\mathbf{r}_1, \ldots, \mathbf{r}_{k_{\mathsf{ARE}}} \in \{0,1\}^{n'_{\mathsf{ARE}} - n_{\mathsf{ARE}}}$ used for running $\mathsf{ARE.Encode}(\cdot, \mathbf{x}, \mathbf{r})$. Set $\mathbf{a}_i = (\mathbf{x}, \mathbf{r}_i)$ for $i \in [k_{\mathsf{ARE}}]$. Here $\mathbf{a}_i \in \{0,1\}^{n'_{\mathsf{ARE}} = n_{\mathsf{PPE}}}$.
- Compute $(\mathsf{P}, \mathsf{S}) \leftarrow \mathsf{PPE.PreProc}(1^{n_{\mathsf{PPE}}}, 1^{k_{\mathsf{PPE}}}, p, \mathcal{Q}, \mathbf{a})$. Output $\mathsf{PI} = \mathsf{P}$ and $\mathsf{SI} = \mathsf{S}$.

Encoding $\mathsf{PRE.Encode}(C, (\mathsf{PI}, \mathsf{SI}))$: Run the following steps:

- By the efficiency property of ARE, for any circuit $C \in \mathcal{F}_{\mathsf{ARE}, n_{\mathsf{ARE}}, m_{\mathsf{ARE}}, k_{\mathsf{ARE}}, \lambda}$, for $i \in [T]$ where T is the output length of $\mathsf{ARE.Encode}(C, \cdot)$, the i^{th} output bit of $\mathsf{ARE.Encode}(C, \cdot)$ is computable by an efficiently generatable polynomial $f_i \in \mathcal{F}_{\mathsf{PPE}, d, n_{\mathsf{PPE}}, \mathcal{Q}, k_{\mathsf{PPE}}}$. Let g_{f_i} be the degree $(d', 2)$-polynomial evaluating $\mathsf{PPE.Eval}(f_i, \cdot)$. Compute $y_i = \mathsf{PPE.Eval}(f_i, \mathsf{PI}, \mathsf{SI}) = g_{f_i}(\mathsf{PI}, \mathsf{SI})$. Output $\mathbf{y} = (y_1, \ldots, y_T)$.

Decode $\mathsf{PRE.Decode}(\mathbf{y})$: Run and output $\mathsf{ARE.Decode}(\mathbf{y}) = \mathbf{z}$.

Figure 3.1: The Description of the PRE scheme.

bit b, where as the last hybrid is independent of b. Let C be a circuit in $\mathcal{F}_{\mathsf{PPE}}$ and $\mathbf{x}_0, \mathbf{x}_1$ be inputs such that $C(\mathbf{x}_0) = C(\mathbf{x}_1)$.

Hybrid$_0$: In this hybrid, we compute (P, S) by preprocessing $\mathbf{a} = (\mathbf{a}_1, \ldots, \mathbf{a}_{k_{\mathsf{PRE}}})$. Here each $\mathbf{a}_i = (\mathbf{x}_b, \mathbf{r}_i)$ where $\mathbf{b} \leftarrow \{0, 1\}$. Let $\mathbf{y}$ be the output of $\mathsf{PRE.Encode}$ operation on input $C, \mathsf{P}, \mathsf{S}$. Output of the hybrid is $(\mathsf{P}, \mathbf{y})$.

Hybrid$_1$: In this hybrid, we compute (P, S) by preprocessing $\mathbf{a}$ generated as in the previous hybrid. Let $\mathbf{y}$ be the output of $\mathsf{ARE.Encode}(C, \mathbf{x}, \mathbf{r})$, where $\mathbf{a}_i = (\mathbf{x}_b, \mathbf{r}_i)$. Output of the hybrid is $(\mathsf{P}, \mathbf{y})$.

Note that **Hybrid**$_0$ is statistically close to **Hybrid**$_1$ due to the correctness of PPE scheme. In one case $\mathbf{y}$ is actually an output of $\mathsf{PPE.Eval}$ function performed over (P, S) using function f_i, in the other case, it is generated by computing $y_i = f_i(\mathbf{a})$. The claim thus follows, due to the correctness of PPE scheme.

Hybrid$_2$: In this hybrid, we compute (P, S) by preprocessing the all 0 string. $\mathbf{y}$ is computed as in the previous hybrid. Output of the hybrid is $(\mathsf{P}, \mathbf{y})$.

The above two hybrids are indistinguishable due to the security of the PPE scheme. In one case P is generated by preprocessing $\mathbf{a}$, in the other case it is generated by preprocessing the all 0 string.

Hybrid$_3$: In this hybrid, we compute P as in the previous hybrid. The only change is that $\mathbf{y}$ is computed as $\mathsf{ARE.Encode}(C, \mathbf{x}_0, \mathbf{r})$. Output of the hybrid is $(\mathsf{P}, \mathbf{y})$.

The above two hybrids are indistinguishable due to the security of the ARE scheme. In both the hybrids, P is generated by encoding 0 string. In one case $\mathbf{y}$ is generated by running $\mathbf{y} = \mathsf{ARE.Encode}(C, \mathbf{x}_b; \mathbf{r})$, and in the other, $\mathbf{y} = \mathsf{ARE.Encode}(C, \mathbf{x}_0; \mathbf{r})$. These are indistinguishable due to the security of the ARE scheme. Since the last hybrid is independent of b, the security holds.

Sublinear Efficiency. Let us now find out the time to run $\mathsf{PRE.PreProc}(1^{\lambda}, 1^{n_{\mathsf{PRE}}}, 1^{m_{\mathsf{PRE}}}, 1^{k_{\mathsf{PRE}}}, p, \mathbf{x})$. This algorithm runs in two steps.

1. In the first step it arranges $\mathbf{x}$ and a random vector $\mathbf{r}$ into $\mathbf{a}$,

2. In the second step, it runs $\mathsf{PPE.PreProc}(1^{n_{\mathsf{PPE}}}, 1^{k_{\mathsf{PPE}}}, p, \mathcal{Q}, \mathbf{a})$.

The first step can be implemented by a circuit of size $O(|\mathbf{a}|) = O(n_{\mathsf{PRE}} \cdot k_{\mathsf{PRE}}) = O(n'_{\mathsf{ARE}} \cdot k_{\mathsf{PRE}}) = O(n_{\mathsf{PRE}} \cdot k_{\mathsf{PRE}} \cdot \lambda^{c_1} + {m_{\mathsf{PRE}}}^{1-c_2} \cdot k_{\mathsf{PRE}} \cdot \lambda^{c_1})$. Due to the sublinear efficiency of the PPE scheme, the second step takes:

$$t_{\mathsf{PPE}} = O((n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}^{\epsilon_1} + m_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}^{1-\epsilon_2} + k_{\mathsf{PPE}}^{\epsilon_3})\, \mathsf{poly}(\log_2 p)).$$

for some constants $\epsilon_1, \epsilon_2, \epsilon_3 > 0$ and some polynomial poly. Substituting $n_{\mathsf{PPE}} = n'_{\mathsf{ARE}} = O(n_{\mathsf{PRE}} \cdot \lambda^{c_1} + {m_{\mathsf{PRE}}}^{1-c_2} \cdot \lambda^{c_1})$, $m_{\mathsf{PPE}} = m'_{\mathsf{ARE}} = O((n_{\mathsf{PRE}} + m_{\mathsf{PRE}})\lambda^{c_1})$ and $k_{\mathsf{PRE}} = k_{\mathsf{ARE}}$, we get that:

$$t_{\mathsf{PPE}} = O((n_{\mathsf{PRE}} {k_{\mathsf{PRE}}}^{\max\{\epsilon_1, 1-\epsilon_2\}} + {m_{\mathsf{PRE}}}^{1-c_2} {k_{\mathsf{PRE}}}^{\epsilon_1} + m_{\mathsf{PRE}} \cdot {k_{\mathsf{PRE}}}^{1-\epsilon_2} + {k_{\mathsf{PRE}}}^{\epsilon_3})\, \mathsf{poly}(\log_2 p) \lambda^{c_1}).$$

Thus adding the two times, we get the total time $t_{\mathsf{PRE}} = O(t_{\mathsf{PPE}})$. Thus, PPE satisfies sublinear efficiency.

Complexity Requirement. Observe that $\mathsf{PRE.Encode}(C, (\mathsf{PI}, \mathsf{SI}))$ computes $\mathbf{y} = (g_{f_1}(\mathsf{PI}, \mathsf{SI}), \ldots, g_{f_T}(\mathsf{PI}, \mathsf{SI}))$ where $f_i(\mathbf{a})$ computes the i^{th} bit of $\mathsf{ARE.Encode}(C, \mathbf{x}, \mathbf{r})$. Note that due to the efficiency property of ARE f_i is of the form:

$$f_i(\mathbf{a}) = \sum_{j \in [k_{\mathsf{PRE}}], Q \in \mathcal{Q}} \mu_{i,Q,j} \mathsf{Mon}_Q(\mathbf{a}_j),$$

where $\mu_{i,Q,j} \in \mathbb{Z}$. Thus, $f_i \in \mathcal{F}_{\mathsf{PPE}, d, n_{\mathsf{PPE}}, \mathcal{Q}, k_{\mathsf{PPE}}}$. Due to the complexity requirement of PPE, g_{f_i} is degree $(d', 2)$-polynomial over $(\mathsf{PI}, \mathsf{SI})$. This proves the claim.

Summing up. We sum up with the following theorem:

Theorem 3.1.1. *Assuming the existence of a* PPE *scheme (Definition 3.1.3) and an* ARE *scheme (Definition 3.1.8), then the scheme above is a* PRE *scheme satisfying Definition 2.5.1. Further, if both the underlying primitives are subexponentially secure, then so is the resulting* PRE *scheme.*

In Theorem 4.2.2 it is shown that an ARE scheme satisfying Definition 3.1.8 can be constructed assuming PRG in NC^0 with polynomial stretch (Definition 5.2.1). It is shown in Theorem 4.2.2 that a PPE scheme satisfying Definition 3.1.3 can be constructed assuming the δ-LPN assumption (Definition 4.1.1) for any constant $\delta > 0$. Combining these two theorems we have:

Theorem 3.1.2. *Assume that there exists two constant $\delta, \epsilon > 0$ such that:*

- δ-LPN *assumption (Definition 4.1.1) holds,*
- *There exists a* PRG *in* NC^0 *with a stretch $n^{1+\epsilon}$ where n is the length of the input (Definition 5.2.1),*

Then, there exists a PRE *scheme (Definition 2.5.1). Further, assuming the underlying assumptions are subexponentially secure, then so is the resulting* PRE *scheme.*

CHAPTER 4

Preprocessed Polynomial Encoding

In this chapter, we show how to construct a Preprocessed Polynomial Encoding scheme PPE. Let us first recall the goal.

Preprocessed Polynomial Encoding Consider the following task. Let us suppose that there is a parameter $k_{\mathsf{PPE}} \in \mathbb{N}$, and k_{PPE} inputs $\mathbf{x}_1, \ldots, \mathbf{x}_{k_{\mathsf{PPE}}} \in \{0,1\}^{n_{\mathsf{PPE}}}$ collectively denoted as $\mathbf{x}$. Let us say that we are interested in computing constant degree $d > 2$ polynomials $f(\mathbf{x}) \mod p$ for some prime p where f has the following form:

- There exists a d monomial pattern $\mathcal{Q} = (Q_1, \ldots, Q_{m_{\mathsf{PPE}}})$ over n_{PPE} variables.
- $f(\mathbf{x})$ respects monomial pattern in all variables $\mathbf{x}_1, \ldots, \mathbf{x}_{k_{\mathsf{PPE}}}$. That is,

  $$f(\mathbf{x}) = \sum_{i \in [k_{\mathsf{PPE}}], Q \in \mathcal{Q}} \mu_{Q,i} \mathsf{Mon}_Q(\mathbf{x}_i)$$

  for integers $\mu_{Q,i}$.

The question is then, is it possible to preprocess $\mathbf{x}$ into two vectors (P, S) such that for every such f, there exists efficiently samplable degree $(O(d), 2)$-polynomials g_f (which only depend on f):

$$g_f(\mathsf{P}, \mathsf{S}) = f(\mathbf{x}) \mod p,$$

with overwhelming probability. Along the way, we must also ensure that P does not reveal any information about $\mathbf{x}$ (computationally). And thus, this gives us a way to transfer complexity in the secret input: In the initial model, we compute a degree d polynomial f on the secret input $\mathbf{x}$, whereas now we compute polynomials g_f which is just degree 2 in the secret input and still constant degree overall.

Now make an important observation: Such a model is trivial to realize if there is no restriction on the preprocessing model. Indeed, one can set $\mathsf{P} = \perp$ and S as the precomputation of all degree $\frac{d}{2}$ monomials over $\mathbf{x}$. Such a PPE will be useless for us. In formulating a meaningful requirement we should be mindful of the fact that we are only interested in evaluating polynomials f that take k_{PPE} blocks of input, where every block is evaluated on a fixed set of monomials $\mathcal{Q}$. Thus, in all the total monomials that can every be evaluated are $O(m_{\mathsf{PRE}} \cdot k_{\mathsf{PRE}})$. Therefore we ask that the computation of (P, S) takes asymptotically lesser time than $m_{\mathsf{PRE}} \cdot k_{\mathsf{PRE}}$ as k_{PPE} increases with respect to some efficiency measure. This is where amortization comes into play. We can ask for two flavours of requirement, where the second one is stronger than the first one.

- (Sublinear size) Here, we require that the size of (P, S) is sublinear in $m_{\mathsf{PRE}} \cdot k_{\mathsf{PRE}}$,
- (Sublinear preprocessing time) Here, we require that the time it takes to compute (P, S) is sublinear in $m_{\mathsf{PRE}} \cdot k_{\mathsf{PRE}}$,

By being sublinear, we really mean that there exist some constant $c_1, c_2 > 0$ such that the size/time is bounded by $O((n_{\mathsf{PPE}} k_{\mathsf{PPE}}^{c_1} + m_{\mathsf{PPE}} k_{\mathsf{PPE}}^{1-c_2})\,\mathsf{poly}(\log_2 p))$ for some polynomial poly. The reason this is called sublinear is that when $m_{\mathsf{PPE}} = n_{\mathsf{PPE}}^{1+\epsilon}$ for some $\epsilon > 0$, then we can find a $k_{\mathsf{PPE}} = n_{\mathsf{PPE}}^{\Omega(1)}$ such that the value is $O((m_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}})^{1-\gamma}\,\mathsf{poly}(\log_2 p))$ for some constant $\gamma > 0$.

In what follows, we show how to construct such a notion. In this thesis, we will construct the PPE scheme that satisfies the stronger second property. Note that, to construct $i\mathcal{O}$ from well-studied assumptions, it would have actually sufficed to construct the first notion at the cost of making an additional assumption of Learning with Errors [Reg05]. This was actually implicitly done in [JLS21a]. We depart from the presentation in [JLS21a] and construct a scheme satisfying the second property directly, based on [JLS21b]. Next, we first give an overview of the construction. In the overview we first argue that the first property is satisfied. Then, we take a close look and show that the scheme also satisfies the first property.

4.1 Overview of the Construction

A first observation to build such a PPE scheme is that allowing a public input P is really crucial for building a PPE. If it was not allowed, building a PPE would mean that we can generically convert polynomials of degree d into degree polynomials of degree 2 (in the preprocessing model) which is something that we should not expect for general polynomials. Also intuitively, P computationally hides or "encrypts" $\mathbf{x}$. Thus, as a first attempt, we will simply encrypt $\mathbf{x}$ using the LPN assumption, and set the encryption as P. Let us recall the LPN assumption:

The LPN Assumption: Let p be any prime modulus. We define the distribution $\mathcal{D}_r(p)$ as the distribution that outputs 0 with probability $1-r$ and a random element from $\mathbb{Z}_p$ with the remaining probability. Below we define δ-LPN Assumption [BFKL94, IPS08, AAB15, BCGI18].

Definition 4.1.1 (δ-LPN Assumption, [BFKL94, IPS08, AAB15, BCGI18]). *Let $\delta \in (0,1)$. We say that the δ-LPN Assumption is true if the following holds: For any constant $\eta_p > 0$, any function $p : \mathbb{N} \to \mathbb{N}$ s.t., for every $\ell \in \mathbb{N}$, $p(\ell)$ is a prime of ℓ^{η_p} bits, any constant $\eta_n > 0$, we set $p = p(\ell)$, $n = n(\ell) = \ell^{\eta_n}$, and $r = r(\ell) = \ell^{-\delta}$, and we require that the following two distributions are computationally indistinguishable:*

$$\left\{ (\mathbf{A}, \mathbf{b} = \mathbf{s} \cdot \mathbf{A} + \mathbf{e}) \;\mid\; \mathbf{A} \leftarrow \mathbb{Z}_p^{\ell \times n},\ \mathbf{s} \leftarrow \mathbb{Z}_p^{1 \times \ell},\ \mathbf{e} \leftarrow \mathcal{D}_r^{1 \times n}(p) \right\}_{\ell \in \mathbb{N}}$$

$$\left\{ (\mathbf{A}, \mathbf{u}) \;\mid\; \mathbf{A} \leftarrow \mathbb{Z}_p^{\ell \times n},\ \mathbf{u} \leftarrow \mathbb{Z}_p^{1 \times n} \right\}_{\ell \in \mathbb{N}}$$

In addition, we say that subexponential δ-LPN holds if the two distributions above are are subexponentially indistinguishable.

Encrypting $\mathbf{x}$ using LPN assumption. In order to encrypt $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_{k_{\mathsf{PPE}}}) \in \{0,1\}^{n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}}$, we simply generate $n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}$ δ-LPN samples over $\mathbb{Z}_p$ for the constant $\delta > 0$, for which the assumption holds.

- Sample coefficient vectors as follows. For $i \in [k_{\mathsf{PPE}}],\ j \in [n_{\mathsf{PPE}}]$, sample $\mathbf{a}_{i,j} \leftarrow \mathbb{Z}_p^{k_{\mathsf{PPE}}}$.
- Sample secret $\mathbf{s} \leftarrow \mathbb{Z}_p^{k_{\mathsf{PPE}}}$.
- Sample sparse errors and generate LPN samples as follows. Sample $e_{i,j} \leftarrow \mathcal{D}_{k_{\mathsf{PPE}}^{-\delta}}(p)$ for $i \in [k_{\mathsf{PPE}}],\ j \in [n_{\mathsf{PPE}}]$. Compute $d_{i,j} = \langle \mathbf{a}_{i,j}, \mathbf{s} \rangle + e_{i,j} \mod p$.
- Compute encryption of $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_{k_{\mathsf{PPE}}})$ as follows. For $i \in [k_{\mathsf{PRE}}]$, let $\mathbf{x}_i = (x_{i,1}, \ldots, x_{i,n_{\mathsf{PRE}}})$. Compute $b_{i,j} = d_{i,j} + x_{i,j} \mod p = \langle \mathbf{a}_{i,j}, \mathbf{s} \rangle + e_{i,j} + x_{i,j} \mod p$.
- Set $\mathsf{P} = \{\mathbf{a}_{i,j}, b_{i,j}\}_{i \in [k_{\mathsf{PRE}}], j \in [n_{\mathsf{PRE}}]}$.

Observe that P contains $O(n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}^2)$ field elements, and so it does not violate the sublinearity constraints.

How to Set S? Observe that it suffices to compute $\mathsf{Mon}_{Q_j}(\mathbf{x}_i)$ for every $i \in [k_{\mathsf{PPE}}],\ j \in [m_{\mathsf{PPE}}]$. Thus, our initial attempt is to set $\mathsf{S}_0 = (1, \mathbf{s})^{\otimes \lceil \frac{d}{2} \rceil}$. The reason for this is that it lets one compute:

$$\begin{aligned} &\mathsf{Mon}_{Q_j}(b_{i,1} - \langle \mathbf{a}_{i,1}, \mathbf{s} \rangle, \ldots, b_{i,n_{\mathsf{PPE}}} - \langle \mathbf{a}_{i,n_{\mathsf{PPE}}}, \mathbf{s} \rangle) \\ &= \mathsf{Mon}_{Q_j}(x_{i,1} + e_{i,1}, \ldots, x_{i,n_{\mathsf{PPE}}} + e_{i,n_{\mathsf{PPE}}}) \mod p, \end{aligned}$$

This polynomial is degree d in P, and degree d in $\mathbf{s}$. Since $\mathsf{S}_0 = (1, \mathbf{s})^{\otimes \lceil \frac{d}{2} \rceil}$ it is just degree two in S_0. Furthermore, S_0 has $O(k_{\mathsf{PPE}}^{\lceil \frac{d}{2} \rceil})$ field elements, and thus it does not violate the sublinearity constraint. But most importantly, observe that:

$$\Pr[\mathsf{Mon}_{Q_j}(x_{i,1} + e_{i,1}, \ldots, x_{i,n_{\mathsf{PPE}}} + e_{i,n_{\mathsf{PPE}}}) \neq \mathsf{Mon}_{Q_j}(\mathbf{x}_i)] \leq \frac{d}{k_{\mathsf{PPE}}^{\delta}},$$

because the probability of error $e_{i,j} \neq 0$ is $k_{\mathsf{PPE}}^{-\delta}$. Further, for every set $Q_j \in \mathcal{Q}$, $O(k_{\mathsf{PPE}}^{1-\delta})$ monomials are not correctly evaluated this way in expectation, but $k_{\mathsf{PPE}} - O(k_{\mathsf{PPE}}^{1-\delta})$ are! This means that setting S as S_0 almost solves the problem. However, we still need a way to correct $O(m_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}^{1-\delta})$ monomials. Our approach is therefore to come up with another component S_1, such that this correction will be done by quadratic polynoials in S_1. Further, the size of S_1 will follow the sublinearity rule.

Correcting Monomials. Our next observation is that for each of the m_{PPE} sets $Q_j \in \mathcal{Q}$, out of k_{PPE} monomials $\{\mathsf{Mon}_{Q_j}(\mathbf{x}_i)\}_{i\in[k_{\mathsf{PPE}}]}$, $O(k_{\mathsf{PPE}}^{1-\delta})$ need correction. In fact, since the errors used in LPN samples corresponding to k_{PRE} monomials are independent, such monomials exhibit tight concentration due to Chernoff bound. With overwhelming probability the number of such monomials that need to be corrected are $O(k_{\mathsf{PPE}}^{1-\delta})$. Thus, our approach is the following. We set $\mathsf{S}_1 = (\mathsf{S}_{1,1}, \ldots, \mathsf{S}_{1,m_{\mathsf{PPE}}})$ where $\mathsf{S}_{1,j}$ "compress" corrections for $\{\mathsf{Mon}_{Q_j}(\mathbf{x}_i)\}_{i\in[k_{\mathsf{PPE}}]}$ such that the decompression is a quadratic function.

The rough idea is that for every Q_j we set up matrix $\mathbf{M}_j$ of dimension $[\sqrt{k_{\mathsf{PPE}}}] \times [\sqrt{k_{\mathsf{PPE}}}]$ (thus having k_{PPE} entries). Define $\mathsf{Corr}_{j,i} = \mathsf{Mon}_{Q_j}(\mathbf{x}_i) - \mathsf{Mon}_{Q_j}(\mathbf{x}_i + \mathbf{e}_i)$ for all $i \in [k_{\mathsf{PPE}}]$ where $\mathbf{e}_i = (e_{i,1} \ldots, e_{i,n_{\mathsf{PPE}}})$. We simply arrange Corr_j within the matrix $\mathbf{M}_j$ using any public canonical map. For example, one such map can be dividing an index $i \in k_{\mathsf{PPE}}$, by $\sqrt{k_{\mathsf{PPE}}}$ and then using its remainder and the quotient as the map into $\mathbf{M}_j$.

Now observe that with overwhelming probability, Corr_j is sparse with just $O(k_{\mathsf{PPE}}^{1-\delta})$ non-zero entries. Therefore, the matrix is also sparse with $O(k_{\mathsf{PPE}}^{1-\delta})$ non-zero entries. Thus, we simply partition $\mathbf{M}_j$ into $t_1 = k_{\mathsf{PPE}}^{1-\delta}$ sub-matrices, $\mathbf{M}_{j,\ell}$ of dimension $[k_{\mathsf{PPE}}^{\frac{\delta}{2}}] \times [k_{\mathsf{PPE}}^{\frac{\delta}{2}}]$ for $\ell \in [t_1]$. The intention of doing this is that in expectation each sub-matrix will have at most a constant number of non-zero entries, and therefore each sub-matrix is low rank. In fact, using a concentration argument, we can prove that with overwhelming probability each matrix has at most $t_2 = k_{\mathsf{PPE}}^{\rho}$ non-zero entries for a really small constant $\rho \ll \delta$. Thus, we can find $\mathbf{U}_{j,\ell}, \mathbf{V}_{j,\ell} \in \mathbb{Z}_p^{k_{\mathsf{PPE}}^{\frac{\delta}{2}} \times t_2}$ such that $\mathbf{M}_{j,\ell} = \mathbf{U}_{j,\ell}\mathbf{V}_{j,\ell}^{\top}$ using low rank matrix decomposition methods. We set $\mathsf{S}_j = \{\mathbf{U}_{j,\ell}, \mathbf{V}_{j,\ell}\}_{\ell \in [t_1]}$. Observe that S_j contains $2 \cdot t_1 \cdot k_{\mathsf{PPE}}^{\frac{\delta}{2}} \cdot t_2 = O(k_{\mathsf{PPE}}^{1-\frac{\delta}{2}+\rho})$ field elements, which is under our permitted bound as $\rho \ll \delta$.

Functionality. Once we have set $\mathsf{S}_0 = (1, \mathbf{s})^{\otimes \lceil \frac{d}{2} \rceil}$ and $\mathsf{S}_1 = (\mathsf{S}_{1,1}, \ldots, \mathsf{S}_{1,m_{\mathsf{PPE}}})$, we can compute $\mathsf{Mon}_{Q_j}(\mathbf{x}_i)$ using a degree $(d, 2)$-polynomial as follows. Let us assume that the correction for this monomial is embedded in $\mathbf{M}_{j,\ell_1}[\ell_2, \ell_3]$, parse $\mathsf{S}_{1,j} = \{\mathbf{U}_{j,\ell}, \mathbf{V}_{j,\ell}\}_{\ell \in [t_1]}$. Then compute:

$$\begin{aligned}
&\mathsf{Mon}_{Q_j}(b_{i,1} - \langle \mathbf{a}_{i,1}, \mathbf{s} \rangle, \ldots, b_{i,1} - \langle \mathbf{a}_{i,n_{\mathsf{PPE}}}, \mathbf{s} \rangle) + \mathbf{U}_{j,\ell_1}\mathbf{V}_{j,\ell_1}^{\top}[\ell_2, \ell_3] \\
&= \mathsf{Mon}_{Q_j}(\mathbf{x}_i + \mathbf{e}_i) + \mathbf{U}_{j,\ell_1}\mathbf{V}_{j,\ell_1}^{\top}[\ell_2, \ell_3] \\
&= \mathsf{Mon}_{Q_j}(\mathbf{x}_i + \mathbf{e}_i) + \mathsf{Mon}_{Q_j}(\mathbf{x}_i) - \mathsf{Mon}_{Q_j}(\mathbf{x}_i + \mathbf{e}_i) \\
&= \mathsf{Mon}_{Q_j}(\mathbf{x}_i)
\end{aligned}$$

Security. Observe that even given $\mathbf{x}$, P is indistinguishable to random field elements over $\mathbb{Z}_p$, due to the LPN security, so security follows almost immediately.

Sublinear Time. In the above discussion, we argued the sublinearity of the size of P, S. It should not be clear at all, why it should have a sublinear size circuit. Nevertheless, this is the case and we show this in Section 4.2.1. This argument gives a concrete implementation of the preprocessing algrorithm.

4.2 PPE Construction Details

In this section, we present our construction of PPE scheme. Before delving into the construction, we describe the list of notations that will be useful:

- Parameters $t_1 = \lceil k^{1-\delta} \rceil$ and $T = \lceil k^{\delta/2} \rceil$. Observe that $2 \cdot k_{\mathsf{PPE}} \geq t_1 \cdot T^2 \geq k_{\mathsf{PPE}}$.
- t_2 is the slack parameter. It is set as $k_{\mathsf{PPE}}^{\frac{\delta}{10}}$,

- **Map ϕ:** We define an injective map ϕ which canonically maps k_{PPE} elements into t_1 buckets (equivalently called as a matrices in the text below), each having a size of $T \times T$. For every $j \in [k_{\mathsf{PPE}}]$, $\phi(j) = (j_1, (j_2, j_3))$ where $j_1 \in [t_1]$, $(j_2, j_3) \in [T] \times [T]$. Such a map can be computed in time polynomial in $\log_2 k_{\mathsf{PPE}}$ and can be computed by first dividing $j \in [k_{\mathsf{PPE}}]$ by t_1 and setting its remainder as j_1. Then the quotient of this division is further divided by T. The quotient and the remainder of this division are set as (j_2, j_3).

Construction of PPE

$(\mathsf{P}, \mathsf{S}) \leftarrow \mathsf{PreProc}(1^{n_{\mathsf{PPE}}}, 1^{k_{\mathsf{PPE}}}, p, \mathcal{Q} = (Q_1, \ldots, Q_{m_{\mathsf{PPE}}}), \mathbf{x})$**:** Below we describe the pseudo-code. We show how to construct a circuit for the same when we talk about pre-processing efficiency property of the scheme. Perform the following steps:

- Parse $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_{k_{\mathsf{PPE}}})$ where each $\mathbf{x}_j \in \mathbb{Z}_p^{n_{\mathsf{PPE}}}$. Parse $\mathbf{x}_j = (x_{j,1}, \ldots, x_{j,n_{\mathsf{PPE}}})$.
- The overall outline is the following: We first show how to sample components $\mathsf{P}' = (\mathsf{P}_1, \ldots, \mathsf{P}_{k_{\mathsf{PPE}}})$, and then how to sample S along with a boolean variable flag. P will be set as $(\mathsf{flag}, \mathsf{P}')$.
- **Sampling $\mathsf{P}' = (\mathsf{P}_1, \ldots, \mathsf{P}_{k_{\mathsf{PPE}}})$:** Sample $\mathbf{s} \leftarrow \mathbb{Z}_p^{k_{\mathsf{PPE}}}$. For every $i \in [n_{\mathsf{PPE}}]$, and $j \in [k_{\mathsf{PPE}}]$:

  1. Sample $\mathbf{a}_{j,i} \leftarrow \mathbb{Z}_p^{k_{\mathsf{PPE}}}$.
  2. Sample $e_{j,i} \leftarrow \mathsf{Ber}(k_{\mathsf{PPE}}^{-\delta}) \cdot \mathbb{Z}_p$. Denote $\mathbf{e}_j = (e_{j,1}, \ldots, e_{j,n_{\mathsf{PPE}}})$.
  3. Compute $b_{j,i} = \langle \mathbf{a}_{j,i}, \mathbf{s} \rangle + e_{j,i} + x_{j,i} \mod p$.

  For $j \in [k_{\mathsf{PPE}}]$, set $\mathsf{P}_j = \{\mathbf{a}_{j,i}, b_{j,i}\}_{i \in [n_{\mathsf{PPE}}]}$.
- **Sampling S:** S has $m_{\mathsf{PPE}} + 1$ components. That is, $\mathsf{S} = (\mathsf{S}_0, \ldots, \mathsf{S}_{m_{\mathsf{PPE}}})$. Set $\mathsf{S}_0 = (1, \mathbf{s})^{\otimes \lceil \frac{d}{2} \rceil}$. We now show how to compute S_r for $r \in [m_{\mathsf{PPE}}]$.

  1. For $j \in [k_{\mathsf{PPE}}]$, compute $\mathsf{Corr}_{r,j} = \mathsf{Mon}_{Q_r}(\mathbf{x}_j) - \mathsf{Mon}_{Q_r}(\mathbf{x}_j + \mathbf{e}_j)$.

2. Initialize for every $\gamma \in [t_1]$, matrices $\mathbf{M}_{r,\gamma}$ in $\mathbb{Z}_p^{T\times T}$ with zero entries.

3. For $j \in [k_{\mathsf{PPE}}]$, compute $\phi(j) = (j_1, (j_2, j_3))$ and set $\mathbf{M}_{r,j_1}[j_2, j_3] = \mathsf{Corr}_{r,j}$. If any matrix $\mathbf{M}_{r,\gamma}$ for $\gamma \in [t_1]$, has more than t_2 non-zero entries, then set $\mathsf{flag}_r = 0$. Otherwise, set $\mathsf{flag}_r = 1$.

4. If $\mathsf{flag}_r = 1$, then, for $\gamma \in [t_1]$, compute matrices $\mathbf{U}_{r,\gamma}, \mathbf{V}_{r,\gamma}^\top \in \mathbb{Z}_p^{T\times t_2}$ such that $\mathbf{M}_{r,\gamma} = \mathbf{U}_{r,\gamma} \cdot \mathbf{V}_{r,\gamma}$. Otherwise for every $\gamma \in [t_1]$, set $\mathbf{U}_{r,\gamma}, \mathbf{V}_{r,\gamma}$ to be matrices with zero-entries.

5. Set $\mathsf{S}_r = \{\mathbf{U}_{r,\gamma}, \mathbf{V}_{r,\gamma}\}_{\gamma\in[t_1]}$.

- **Sampling** flag**:** For every $i \in [n_{\mathsf{PPE}}]$, let $\mathsf{Set}_i = \{j \in [k_{\mathsf{PPE}}] | e_{j,i} \neq 0\}$. If any of these sets have size outside the range $[k_{\mathsf{PPE}}^{1-\delta} - t_2 k_{\mathsf{PPE}}^{\frac{1-\delta}{2}}, k_{\mathsf{PPE}}^{1-\delta} + t_2 k_{\mathsf{PPE}}^{\frac{1-\delta}{2}}]$, set $\mathsf{flag} = 0$. Otherwise, set $\mathsf{flag} = \min\{\mathsf{flag}_r\}_{r\in[m]}$.

$y \leftarrow \mathsf{Eval}(f, (\mathsf{P}, \mathsf{S}))$: Parse $\mathsf{P} = (\mathsf{flag}, \mathsf{P}_1, \ldots, \mathsf{P}_{k_{\mathsf{PPE}}})$ where $\mathsf{P}_j = \{\mathbf{a}_{j,i}, b_{j,i}\}_{i\in[n_{\mathsf{PPE}}]}$. Similarly, parse $\mathsf{S} = (\mathsf{S}_0, \ldots, \mathsf{S}_{m_{\mathsf{PPE}}})$. Here $\mathsf{S}_0 = (1, \mathbf{s})^{\otimes\lceil\frac{d}{2}\rceil}$ and $\mathsf{S}_r = \{\mathbf{U}_{r,\gamma}, \mathbf{V}_{r,\gamma}\}_{\gamma\in[t_1]}$ for $r \in [m_{\mathsf{PPE}}]$. Parse $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_{k_{\mathsf{PPE}}})$ and $f(\mathbf{x}) = \sum_{r\in[m_{\mathsf{PPE}}], j\in[k_{\mathsf{PPE}}]} \mu_{r,j} \mathsf{Mon}_{Q_r}(\mathbf{x}_j)$ for $\mu_{r,j} \in \mathbb{Z}$. Output:

$$g_{f,\mathcal{Q}}(\mathsf{P}, \mathsf{S}) = \sum_{r\in[m_{\mathsf{PPE}}], j\in[k_{\mathsf{PPE}}]} \mu_{r,j} w_{r,j}(\mathsf{P}, \mathsf{S}),$$

where the polynomial $w_{r,j}(\mathsf{P}, \mathsf{S})$ is the following:

$$w_{r,j}(\mathsf{P}, \mathsf{S}) = \mathsf{flag} \cdot \left(\mathsf{Mon}_{Q_r}(b_{j,1} - \langle \mathbf{a}_{j,1}, \mathbf{s}\rangle, \ldots, b_{j,n_{\mathsf{PPE}}} - \langle \mathbf{a}_{j,n_{\mathsf{PPE}}}, \mathbf{s}\rangle) + \mathbf{U}_{r,j_1} \cdot \mathbf{V}_{r,j_1}[j_2, j_3]\right),$$

where $\phi(j) = (j_1, (j_2, j_3))$. We remark that the polynomial above is written as a function of $\mathbf{s}$ and not S_0, however, since we always mean $\mathsf{S}_0 = (1, \mathbf{s})^{\otimes\lceil\frac{d}{2}\rceil}$, we treat this polynomial as some degree-2 polynomial in S_0.

Remark 4.2.1. The only difference to the scheme described in the overview is that the scheme also uses a boolean variable flag. flag will be 1 with overwhelming probability, and is set to 0 when "certain" low probability events happen. As described earlier, the size of the input (P, S) is already sublinear. Later, we describe how even the time to compute it is sublinear.

We now argue various properties involved.

Complexity of Evaluation: Observe that $\mathsf{Eval}(f, (\mathsf{P}, \mathsf{S}))$ is already in the required form, in that, it is computed by a polynomial $g_{f,\mathcal{Q}}$ over $\mathbb{Z}_p$. The only thing that needs to be proved is that the polynomial $g_{f,\mathcal{Q}}$ is degree 2 in S, and $O(d)$ in P. Since $g_{f,\mathcal{Q}}$ is a linear combination of $\{w_{r,j}\}_{j \in [k_{\mathsf{PPE}}], r \in [m_{\mathsf{PPE}}]}$, it suffices to make the claim for the polynomials $w_{r,j}$. The polynomial $w_{r,j}$ has two terms. We analyse both the two terms separately.

- Observe that the second term is degree 2 in S as it is degree 2 in matrices $\{\mathbf{U}_{r,\gamma}, \mathbf{V}_{r,\gamma}\}_{\gamma \in [t_1]}$. The degree of the second term in P is one, as it is degree one in flag.
- The first term is at most degree $d+1$ in P as it is degree one in flag and at most degree d in $\mathbf{a}_{j,i}$ and $b_{j,i}$ variables. It is also at most degree d in $\mathbf{s}$, however we interpret $\mathsf{S}_0 = (1, \mathbf{s})^{\otimes \lceil \frac{d}{2} \rceil}$. Therefore, when written as a polynomial in S_0, its degree is 2.

This proves that the polynomial is at most degree $d+1$ in P and degree 2 in S. We now prove a lemma that will be helpful in arguing the properties below:

Lemma 4.2.1. *Let $d > 0$ be a constant integer, $k_{\mathsf{PPE}} \in \mathbb{N}$, $n_{\mathsf{PPE}} = k_{\mathsf{PPE}}^{O(1)}$, $\mathcal{Q}$ be a d-monomial pattern over n_{PPE} variables, p be any prime, then for any $\mathbf{x} \in \mathbb{Z}_p^{n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}}$, when computing $(\mathsf{P}, \mathsf{S}) \leftarrow \mathsf{PreProc}(1^{n_{\mathsf{PPE}}}, 1^{k_{\mathsf{PPE}}}, p, \mathcal{Q}, \mathbf{x})$:*

$$\Pr[\mathsf{flag} = 1] \geq 1 - \exp(-k_{\mathsf{PPE}}^{\Omega(1)}).$$

Proof. First, we bound the probability $\mathsf{flag}_r = 0$ for $r \in [m_{\mathsf{PPE}}]$, which is done in the third step in the part where one samples S.

Claim 4.2.1. *For every $r \in [m_{\mathsf{PPE}}]$, $\Pr[\mathsf{flag}_r = 0] = O(\exp(-k_{\mathsf{PPE}}^{\Omega(1)}))$.*

Proof. Note that $\mathsf{flag}_r = 0$, when, $\mathbf{M}_{r,\gamma}$ for some $\gamma \in [t_1]$ has more than t_2 non zero entries. The number of entries in $\mathbf{M}_{r,\gamma}$ is bounded by T^2. The probability that any given entry is non-zero is upper bounded by $\frac{d}{k_{\mathsf{PPE}}^{\delta}}$. Since the values $\mathsf{Corr}_{r,j}$ for different values of $j \in [k_{\mathsf{PPE}}]$ are independent, the probability that $\mathbf{M}_{r,\gamma}$ for any given $\gamma \in [t_1]$ has more than t_2 non-zero entries is at most:

$$\begin{aligned}
&= \binom{T^2}{t_2} \frac{d^{t_2}}{k_{\mathsf{PPE}}^{\delta \cdot t_2}} && \text{(Selecting } t_2 \text{ out of } T^2 \text{ locations)} \\
&\leq \frac{(d \cdot e)^{t_2} \cdot T^{2t_2}}{(k_{\mathsf{PPE}}^{\delta} t_2)^{t_2}} && \text{(by Sterling's approximation)} \\
&\leq O(\exp(-k_{\mathsf{PPE}}^{\Omega(1)})) && (\ t_2 = k_{\mathsf{PPE}}^{\delta/10} \text{ and } \delta > 0 \text{ is a constant})\ .
\end{aligned}$$

Taking a union bound for $\gamma \in [t_1]$, we get $\Pr[\mathsf{flag}_r = 0] = O(\exp(-k_{\mathsf{PPE}}^{\Omega(1)}))$. $\square$

Another, condition that affects the setting of the flag is when the size of Set_i is not within $[k_{\mathsf{PPE}}^{1-\delta} - t_2 \cdot k_{\mathsf{PPE}}^{\frac{1-\delta}{2}}, k_{\mathsf{PPE}}^{1-\delta} + t_2 \cdot k_{\mathsf{PPE}}^{\frac{1-\delta}{2}}]$.

Claim 4.2.2. *For any $i \in [n_{\mathsf{PPE}}]$, $\Pr\left[|\mathsf{Set}_i| \notin [k_{\mathsf{PPE}}^{1-\delta} - t_2 \cdot k_{\mathsf{PPE}}^{\frac{1-\delta}{2}}, k_{\mathsf{PPE}}^{1-\delta} + t_2 \cdot k_{\mathsf{PPE}}^{\frac{1-\delta}{2}}]\right] = O(\exp(-k_{\mathsf{PPE}}^{\Omega(1)}))$.*

Proof. This is a straigtforward application of the Chernoff bound. We now recall the Chernoff bound which says the following. Let $\{X_i\}_{i \in \mathbb{N}} \in \{0,1\}$ be iid random variables. Let $N \in \mathbb{N}$, $X = \sum_{i \in [N]} X_i$, $\eta = \mathbb{E}[X]$. Then for any $\rho \in (0,1)$:

$$\Pr[|X - \eta| > \rho\eta] \leq 2\exp\left(-\frac{\rho^2 \cdot \eta}{3}\right).$$

We can compute the required probability as follows. For $j \in [k_{\mathsf{PPE}}]$, define X_j to be 1 with probability $k_{\mathsf{PPE}}^{-\delta}$. Here X_j denotes the event that $j \in \mathsf{Set}_i$. Thus $\mu = k_{\mathsf{PPE}}^{1-\delta}$. Setting $\rho = \frac{t_2}{k_{\mathsf{PPE}}^{\frac{1-\delta}{2}}}$ and plugging into the bound gives us the required probability. This comes up to $2\exp(-\frac{t_2^2}{2})$, which is $O(\exp(-k_{\mathsf{PPE}}^{\Omega(1)}))$. $\square$

Since m_{PPE} and n_{PPE} are polynomials in k_{PPE} doing a union bound over these probabilities, we get that $\Pr[\mathsf{flag} = 0] = O(\exp(-k_{\mathsf{PPE}}^{\Omega(1)}))$. $\square$

Security: We now argue security.

Lemma 4.2.2. *If δ-*LPN *holds, then, the* PPE *scheme described above is secure as per Definition 3.1.3. If the assumption is subexponentially secure then the scheme is also subexponentially secure.*

Proof. We first recall the distribution of the public part of the preprocessed input P. Then, we show indistinguishable hybrids and argue indistinguishability between them. The first hybrid corresponds to the case when P is generated by preprocessing $\mathbf{x} \in \mathbb{Z}_p^{n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}}$, and the last hybrid is independent of $\mathbf{x}$. Recall,

$$\mathsf{P} = (\mathsf{flag} \in \{0,1\}, \mathsf{P}_1, \ldots, \mathsf{P}_{k_{\mathsf{PPE}}}) \qquad \text{where for every } j \in [k_{\mathsf{PPE}}],$$
$$\mathsf{P}_j = \{\mathbf{a}_{j,i}, b_{j,i} = \langle \mathbf{a}_{j,i}, \mathbf{s} \rangle + e_{j,i} + x_{j,i} \mod p\}_{i \in [n_{\mathsf{PPE}}]}.$$

$\mathbf{Hybrid}_0$: This hybrid consists of P above, where it is generated honestly while preprocessing input $\mathbf{x}$.

$\mathbf{Hybrid}_1$: This hybrid is the same as above except that instead of sampling flag honestly, it is always set to 1.

$\mathbf{Hybrid}_2$: This hybrid is the same as above except that for $j \in [k_{\mathsf{PPE}}]$ and $i \in [n_{\mathsf{PPE}}]$, $b_{j,i}$ is sampled uniformly from $\mathbb{Z}_p$.

Due to Lemma 4.2.1, $\mathbf{Hybrid}_0$ and $\mathbf{Hybrid}_1$ are statistically close with statistical distance bounded by $O(\exp(-k^{\Omega(1)}))$.
Then, observe that the only difference between $\mathbf{Hybrid}_1$ and $\mathbf{Hybrid}_2$ is how $\{\mathbf{a}_{j,i}, b_{j,i}\}_{j \in [k_{\mathsf{PPE}}], i \in [n_{\mathsf{PPE}}]}$ is generated. In $\mathbf{Hybrid}_1$, $\mathbf{a}_{j,i}$ is generated as a random vector sampled from $\mathbb{Z}_p^{k_{\mathsf{PPE}}}$. $\{b_{j,i} = \langle \mathbf{a}_{j,i}, \mathbf{s} \rangle + e_{j,i} + x_{j,i} \mod p\}_{j \in [k_{\mathsf{PPE}}], i \in [n_{\mathsf{PPE}}]}$, where $\mathbf{s}$ is also a random vector and $e_{j,i}$ are chosen according to the LPN distribution (with error-rate $k_{\mathsf{PPE}}^{-\delta}$). In $\mathbf{Hybrid}_2$, $b_{j,i}$ is replaced with randomly chosen element. The number of samples released is $n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}$ which is a polynomial in k_{PPE}. Thus, $\mathbf{Hybrid}_1$ and $\mathbf{Hybrid}_2$ are indistinguishable due to δ-LPN. $\square$

Correctness: We now argue correctness:

Lemma 4.2.3. *Let $d > 0$ be an interger constant, $k_{\mathsf{PPE}} \in \mathbb{N}$ and $n_{\mathsf{PPE}} = k_{\mathsf{PPE}}^{\Theta(1)}$, p be any prime, $\mathbf{x} \in \mathbb{Z}_p^{n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}}$. Let $(\mathsf{P}, \mathsf{S}) \leftarrow \mathsf{PreProc}(1^{n_{\mathsf{PPE}}}, 1^{k_{\mathsf{PPE}}}, p, \mathcal{Q}, \mathbf{x})$, then, for any $f \in \mathcal{F}_{\mathsf{PPE},d,n_{\mathsf{PPE}},\mathcal{Q},k_{\mathsf{PPE}}}$:*

$$\Pr[\mathsf{Eval}(f, (\mathsf{P}, \mathsf{S})) \neq f(\mathbf{x})] = O(\exp(-k_{\mathsf{PPE}}^{\Omega(1)})).$$

Proof. Let $f(\mathbf{x}) = \sum_{Q_r \in \mathcal{Q}, j \in [k_{\mathsf{PPE}}]} \mu_{r,j} \cdot \mathsf{Mon}_{Q_r}(\mathbf{x}_j)$ where $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_{k_{\mathsf{PPE}}})$. Note that $\mathsf{Eval}(f, (\mathsf{P}, \mathsf{S})) = g_{f,\mathcal{Q}}(\mathsf{P}, \mathsf{S})$. Now observe that:

$$g_{f,\mathcal{Q}}(\mathsf{P}, \mathsf{S}) = \sum_{Q_r \in \mathcal{Q}, j \in [k_{\mathsf{PPE}}]} \mu_{r,j} w_{r,j}(\mathsf{P}, \mathsf{S}).$$

We will now argue that with probability $1 - O(\exp(-k_{\mathsf{PPE}}^{\Omega(1)}))$, for any $r \in [m_{\mathsf{PPE}}], j \in [k_{\mathsf{PPE}}]$, $w_{r,j}(\mathsf{P}, \mathsf{S}) = \mathsf{Mon}_{Q_r}(\mathbf{x}_j)$ which will complete the proof. Recall that:

$$w_{r,j}(\mathsf{P}, \mathsf{S}) = \mathsf{flag} \cdot \left(\mathsf{Mon}_{Q_r}(b_{j,1} - \langle \mathbf{a}_{j,1}, \mathbf{s} \rangle, \ldots, b_{j,n_{\mathsf{PPE}}} - \langle \mathbf{a}_{j,n_{\mathsf{PPE}}}, \mathbf{s} \rangle) + \mathbf{U}_{r,j_1} \cdot \mathbf{V}_{r,j_1}[j_2, j_3]\right),$$

where $\phi(j) = (j_1, (j_2, j_3)$. As shown in Lemma 4.2.1, $\mathsf{flag} = 1$ with probability $1 - O(\exp(-k_{\mathsf{PPE}}^{\Omega(1)}))$. Thus, with this probability,

$$w_{r,j}(\mathsf{P}, \mathsf{S}) = \mathsf{Mon}_{Q_r}(b_{j,1} - \langle \mathbf{a}_{j,1}, \mathbf{s} \rangle, \ldots, b_{j,n_{\mathsf{PPE}}} - \langle \mathbf{a}_{j,n_{\mathsf{PPE}}}, \mathbf{s} \rangle) + \mathbf{U}_{r,j_1} \cdot \mathbf{V}_{r,j_1}[j_2, j_3].$$

Also observe that:

$$\mathsf{Mon}_{Q_r}(b_{j,1} - \langle \mathbf{a}_{j,1}, \mathbf{s} \rangle, \ldots, b_{j,n_{\mathsf{PPE}}} - \langle \mathbf{a}_{j,n_{\mathsf{PPE}}}, \mathbf{s} \rangle) = \mathsf{Mon}_{Q_r}(\mathbf{x}_j + \mathbf{e}_j),$$

by construction. Finally, observe that when $\mathsf{flag} = 1$, $\mathsf{flag}_r = 1$, and therefore:

$$\mathbf{U}_{r,j_1} \cdot \mathbf{V}_{r,j_1} = \mathbf{M}_{r,j_1},$$

as in the point 4 in the procedure to sample S. Finally, $\mathbf{M}_{r,j_1}[j_2, j_3] = \mathsf{Corr}_{r,j} = \mathsf{Mon}_{Q_r}(\mathbf{x}_j) - \mathsf{Mon}_{Q_r}(\mathbf{x}_j + \mathbf{e}_j)$ as in the point 3 in the procedure to sample S. Thus, if $\mathsf{flag} = 1$ then,

$$\begin{aligned} w_{r,j}(\mathsf{P}, \mathsf{S}) &= \mathsf{Mon}_{Q_r}(\mathbf{x}_j + \mathbf{e}_j) + \mathsf{Corr}_{r,j}, \\ &= \mathsf{Mon}_{Q_r}(\mathbf{x}_j). \end{aligned}$$

This completes the proof. □

4.2.1 Sublinear Time Preprocessing

In the previous section, we argued all the properties except the sublinear efficiency property of the $\mathsf{PreProc}$ algorithm. We discuss this here. Before we proceed, we prove some lemmas about circuit implementability of some programs that will be useful for us in the rest of the section.

4.2.1.1 Useful Lemmas about Circuit Implementability

In this section we recall and prove some results about circuit implementability of certain kinds of programs that will be useful for the rest of the paper. We first recall a result about sorting programs [AKS83]:

Lemma 4.2.4 (Sorting Lemma). *Consider a program $P^{\mathsf{sort}}_{N,B,\phi}$ that takes as input $N \in \mathbb{N}$ strings of size $B \in \mathbb{N}$ bits. It is indexed with a comparator circuit ϕ computing a total ordering defined on two inputs of B bits, that has size T_ϕ. The program outputs the sorted version of the input consisting of N, B bit strings, sorted with respect to ϕ. There exists a family of circuits $\{\mathcal{C}^{\mathsf{sort}}_{N,B,\phi}\}_{N,B,\phi}$, where $\mathcal{C}^{\mathsf{sort}}_{N,B,\phi}$ is efficiently uniformly generatable and has $O(N \cdot B \cdot T_\phi \,\mathsf{poly}(\log(N \cdot B \cdot T_\phi)))$ gates for some polynomial poly.*

We now recall a result from [PF79] which proves that a constant-tape turing machine can be efficiently simulated by a boolean circuit with only poly-logarithmic multiplicative overhead.

Lemma 4.2.5. *For any turing machine M with $O(1)$ tapes running in time $T(n)$ where n is the length of its input, there exists an efficiently computable boolean circuit family $\{\mathcal{C}_n\}_{n\in\mathbb{N}}$ where $\mathcal{C}_n$ takes n bits of input, produces the same output, and has $O(T(n)\,\mathsf{poly}(\log_2 T(n)))$ gates for some polynomial poly.*

We now prove a theorem about programs that makes random access lookup to a database. In order to do so, we first define the notion of a RAM lookup program.

Definition 4.2.1 (RAM lookup program). *A RAM lookup program $P_{q,N}^{\mathsf{lookup}}$ indexed with a number $N \in \mathbb{N}$ and a number $q \in \mathbb{N}$ is a program with the following structure: It takes as input q indices $\{i_1, \ldots, i_q\}$ and a database $\mathsf{DB} \in \{0,1\}^N$ and it outputs $\{\mathsf{DB}[i_1], \ldots, \mathsf{DB}[i_q]\}$.*

We observe the following a really natural statement about such lookup programs, which says that the size of the circuit implementing such a program is almost linear (upto multiplicative polynomial overhead in $\log_2(q \cdot N)$) in q and N.

Lemma 4.2.6. *Let $q, N \in \mathbb{N}$. A RAM lookup program $P_{q,N}^{\mathsf{RAM}}$ can be implemented by an efficiently uniformly generatable boolean circuit of size $O((q+N)\,\mathsf{poly}(\log_2(q \cdot N)))$ for some polynomial poly.*

Proof. We prove this by dividing the task into various (sequential) steps below and then arguing that the steps can be implemented within the required bound.

Step 1: On input $\mathsf{DB} \in \{0,1\}^N$ and locations $\mathbf{y} = (y_1, \ldots, y_q) \in [N]^q$, compute the following set $\mathbf{z}_1 = \{(1, y_1), \ldots, (q, y_q)\}$ that contains q tuples. Namely, append each query with its order.

Step 2: Compute $\mathbf{z}_2$ which is obtained by sorting the tuples inside $\mathbf{z}_1$ in the ascending order according to the second component of the tuples (indices $y_i \in [N]$). This will produce a permutation of tuples in $\mathbf{z}_1$ where the tuples are arranged in increasing order of the query locations in $[N]$.

Step 3: Using $\mathbf{z}_2$ and DB, compute $\mathbf{z}_3$ where for each tuple (i, y_i) in $\mathbf{z}_2$, replace it by $(i, \mathsf{DB}[y_i]) \in [q] \times \{0,1\}$. Call this as $\mathbf{z}_3$.

Step 4: In this step, we sort $\mathbf{z}_3$ according to the first component of the tuples in the increasing order. Remember $\mathbf{z}_3$ had tuples of the form $(i, \mathsf{DB}[y_i])$. Sorting will produce the output $\mathbf{z}_4 = \{(1, \mathsf{DB}[y_1]), \ldots, (q, \mathsf{DB}[y_q])\}$.

Step 5: Finally, process $\mathbf{z}_4$ and remove the first component of the tuple to produce the output $\mathbf{z}_5 = (\mathsf{DB}[y_1], \ldots, \mathsf{DB}[y_q])$. This is the required output.

In the description above, it is immediate by inspection that the program produces the right output. We argue about the size of the circuits implementing each step.
The first and the fifth step can be implemented by a two tape turing machine, making a single pass on the inputs $\mathbf{y}$ and $\mathbf{z}_4$ respectively. Thus from Lemma 4.2.5 they can be implemented by a boolean circuit with size $O(q \cdot \mathsf{poly}(\log(N \cdot q)))$ for some fixed polynomial poly.
The second and the fourth step, can be implemented by a sorting network. Therefore by Lemma 4.2.4, it can be done by a boolean circuit of size $O(q\,\mathsf{poly}(\log q \cdot N))$ for some polynomial poly.
Finally, the third step can also be implemented by a four tape turing machine making a single pass on the input DB and $\mathbf{z}_2$.

We load on the first tape DB and on the second tape, $\mathbf{z}_2$. The third tape will be used to write the output $\mathbf{z}_3$. On the fourth tape, we maintain the counter in $[N]$ of the location where the head on the first tape is currently on. The machine makes a forward pass on the first two tapes while writing on the third tape and updating the counter on the third. It read tuples from $\mathbf{z}_2$, and if the tuple is of the form (i, y_i), it advances its head to the location y_i on the DB tape, and writes $(i, \mathsf{DB}[y]_i)$ on the third tape. The counter tape can be used to navigate to any location on the first tape. Since the input $\mathbf{z}_2$ is sorted with respect to locations, the heads on the first two tape only move in the forward direction. Due to arithmetic operations, it will also have additional multiplicative polynomial overhead in $\log_2(Nq)$ to assist with the navigation. Thus, this turing machine computes the result in $O((q+N) \cdot \mathsf{poly}(\log(q \cdot N)))$ time for some polynomial poly. Due to Lemma 4.2.5, this can be converted to a boolean circuit with size $O((q+N) \cdot \mathsf{poly}(\log(q \cdot N)))$ for another polynomial poly. Thus, combining all these observations, we prove the lemma. $\square$

4.2.1.2 Sublinear Preprocessing Efficiency

Pre-processing Efficiency: We now bound the size of the circuit implementing the $\mathsf{PreProc}$ algorithm.

Theorem 4.2.1. *Let $d > 0$ be an integer constant. $n_{\mathsf{PPE}}, k_{\mathsf{PPE}} \in \mathbb{N}$ be a parameters, $\mathcal{Q}$ be an d-monomial pattern on n_{PPE} variables with m_{PPE} monomials, and p be any prime. The size of the circuit computing $\mathsf{PreProc}(1^{n_{\mathsf{PPE}}}, 1^{k_{\mathsf{PPE}}}, p, \mathcal{Q}, \cdot)$ is $\tilde{O}(k_{\mathsf{PPE}}^{\lceil \frac{d}{2} \rceil} + n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}^2 + m_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}^{1-\frac{2\delta}{5}})$ where $\tilde{O}$ hides multiplicative polynomial factors in $\log_2(p \cdot n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}} \cdot m_{\mathsf{PPE}})$.*

Proof. We will present an explicit circuit implementing the $\mathsf{PreProc}$ algorithm satisfying the claim. In the analysis below we assume that basic field operations $\mod p$ can be implemented by some fixed polynomial in $\log_2 p$. On input $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_{k_{\mathsf{PPE}}}) \in \mathbb{Z}_p^{n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}}$, $\mathsf{PreProc}(1^{n_{\mathsf{PPE}}}, 1^{k_{\mathsf{PPE}}}, p, \mathcal{Q}, \cdot)$ produces the following components:

- $\mathsf{P} = (\mathsf{flag}, \mathsf{P}_1, \ldots, \mathsf{P}_{k_{\mathsf{PPE}}})$.
- $\mathsf{S} = (\mathsf{S}_0, \mathsf{S}_1, \ldots, \mathsf{S}_{m_{\mathsf{PPE}}})$.

The circuit computing the preprocessing is a randomized circuit. It computes the following:

1. Random vectors $\{\mathbf{a}_{j,i}\}_{j \in [k_{\mathsf{PPE}}], i \in [n_{\mathsf{PPE}}]}$ and the secret $\mathbf{s}$ from $\mathbb{Z}_p^{k_{\mathsf{PPE}}}$,
2. Errors $\{e_{j,i}\}_{j \in [k_{\mathsf{PPE}}], i \in [n_{\mathsf{PPE}}]}$ from $\mathsf{Ber}(k_{\mathsf{PPE}}^{-\delta})\mathbb{Z}_p$, and,
3. $\phi(j)$ for all $j \in [k_{\mathsf{PPE}}]$.

The circuit computing all these can be implemented in size $\mathsf{size}_0 = O(n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}^2 \, \mathsf{poly}(\log_2(p \cdot k_{\mathsf{PPE}})))$ for some polynomial poly.

All these computations are fed as input in parallel into three circuits. The first circuit computes $(\mathsf{P}_1, \ldots, \mathsf{P}_{k_{\mathsf{PPE}}})$, the second one computes S_0, and the third one computes $(\mathsf{flag}, \mathsf{S}_1, \ldots, \mathsf{S}_{m_{\mathsf{PPE}}})$. Below we analyze the size of these three circuits: $\mathsf{size}_1,\ \mathsf{size}_2,\ \mathsf{size}_3$. The total size of the circuit computing the preprocessing is therefore: $\mathsf{size}_0 + \mathsf{size}_1 + \mathsf{size}_2 + \mathsf{size}_3$.

Size of the circuit computing $(\mathsf{P}_1, \ldots, \mathsf{P}_{k_{\mathsf{PPE}}})$**:** Observe that P_j is simply $\{\mathbf{a}_{j,i}, \langle \mathbf{a}_{j,i}, \mathbf{s}\rangle + e_{j,i} + x_{j,i} \mod p\}_{i\in[n_{\mathsf{PPE}}]}$. This can be done by a circuit of size $O(n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}} \cdot \mathsf{poly}(\log_2(p \cdot k_{\mathsf{PPE}}))$. Thus, $\mathsf{size}_1 = O(n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}^2 \,\mathsf{poly}(\log_2(p \cdot k_{\mathsf{PPE}})))$ for some polynomial poly.

Size of the circuit computing S_0**:** Note that S_0 consists simply of $(1, \mathbf{s})^{\otimes \lceil \frac{d}{2} \rceil}$. This can be computed by a circuit of size $\mathsf{size}_2 = O(k_{\mathsf{PPE}}^{\lceil \frac{d}{2} \rceil} \,\mathsf{poly}(\log_2(p \cdot k_{\mathsf{PPE}})))$ steps.

The analysis of the third circuit (of size size_3) is somewhat involved and we discuss that next.

Size of the circuit computing $(\mathsf{flag}, \mathsf{S}_1, \ldots, \mathsf{S}_{m_{\mathsf{PPE}}})$**:** In Figure 4.1 we lay down the basic circuit template for the third circuit. We go over each individual circuit, their inputs, outputs and also bound their sizes. The circuit has 4 layers, and each layer has a specific purpose:

Circuit G_1**:**

- The input α_0 to this circuit consists of tuples $\{(i, j, \phi(j) = (j_1, (j_2, j_3)), e_{j,i}, x_{j,i})\}_{i\in[n_{\mathsf{PPE}}], j\in[k_{\mathsf{PPE}}]}$.
- The circuit sorts the input according to the following ordering in the ascending order. To define the comparison, let the tuples be $z = (i, j, (j_1, (j_2, j_3)), e_{j,i}, x_{j,i})$ and $z' = (i', j', (j'_1, (j'_2, j'_3)), e_{j',i'}, x_{j',i'})$.
  1. If $e_{j,i} = 0$ and $e_{j',i'} \neq 0$, output $z > z'$.
  2. If the first rule does not produce a result, check if $i > i'$. If so output $z > z'$.
  3. If both the above criteria does not produce a result, then compare j_1 with j'_1. If $j_1 > j'_1$ output $z > z'$ if $j_1 > j'_1$.
- As a result of this we get α_1 which is a sorted list, where all the tuples z with non zero $e_{j,i}$ come first, and they are sorted in ascending order with respect to index $i \in [n_{\mathsf{PPE}}]$.

Even within those with same value of i they are sorted with respect to j_1.

Size of G_1: Due to Lemma 4.2.4, the circuit can be implemented in size $\mathsf{size}_{G_1} = O(n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}\, \mathsf{poly}(\log_2(p \cdot n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}})))$ for some fixed polynomial poly.

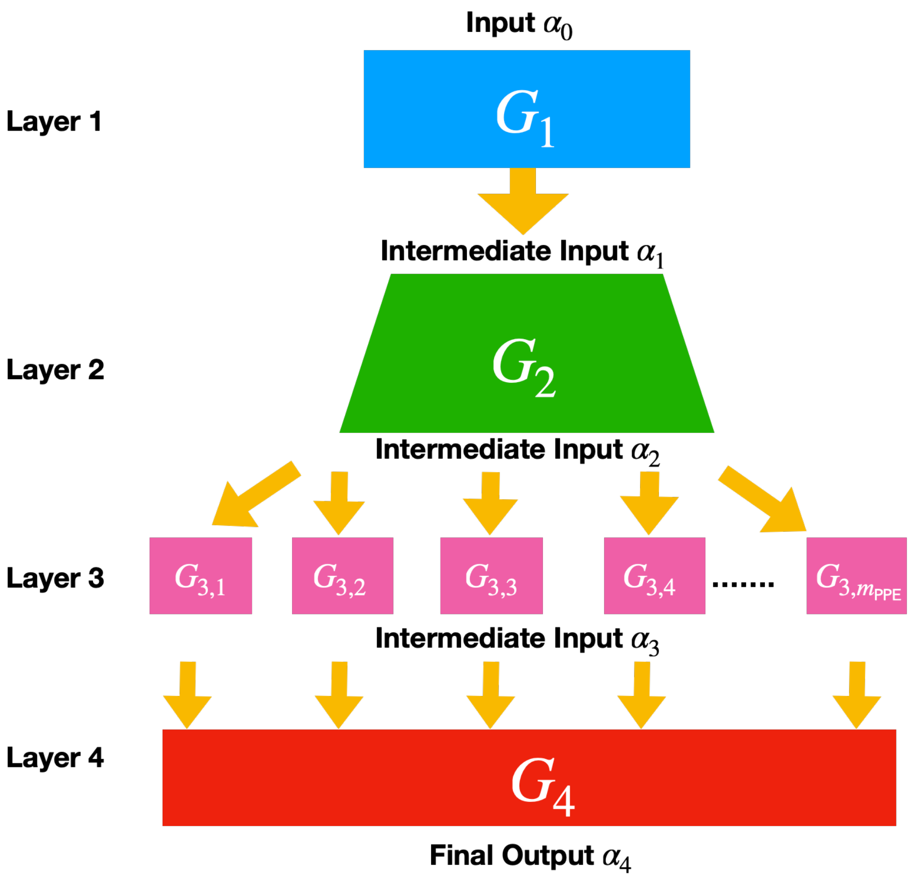

Figure 4.1: Circuit template for the third circuit.

Moving onto the next circuit:

Circuit G_2: The circuit G_2 does two things:

1. First it sets $\mathsf{flag}_{G_2} = 0$ if for any given $i \in [n_{\mathsf{PPE}}]$, the number of non-zero $e_{j,i}$ is not within the range $[k_{\mathsf{PPE}}^{1-\delta} - t_2 \cdot k_{\mathsf{PPE}}^{\frac{1-\delta}{2}}, k_{\mathsf{PPE}}^{1-\delta} + t_2 \cdot k_{\mathsf{PPE}}^{\frac{1-\delta}{2}}]$. Otherwise it is set as 1.

2. The output α_2 consists of flag_{G_2} along with n_{PPE} components, $\alpha_{2,\ell}$ for $\ell \in [n_{\mathsf{PPE}}]$. $\alpha_{2,\ell}$ consists of at most $k_{\mathsf{PPE}}^{1-\delta} + t_2 \cdot k_{\mathsf{PPE}}^{\frac{1-\delta}{2}}$ tuples. These are all tuples that occur in α_1, and are of the form $z = (i, j, \phi(j) = (j_1, (j_2, j_3)), e_{j,i}, x_{j,i})$ where $i = \ell$ and $e_{j,i} \neq 0$. Further, they are sorted with respect to the component j_1. When $\mathsf{flag}_{G_2} \neq 0$, this can always be done. We don't care for its output in the condition when the flag is set to be 0.

Size of G_2: Note that both these steps above can be performed by a two tape Turing machine that keeps α_1 on the first tape. It makes a single pass on the input to compute flag_{G_2}, writes it on the second tape, and then makes another pass to compute tuples $\{\alpha_{2,\ell}\}_{\ell \in [n_{\mathsf{PPE}}]}$. Since α_1 already consists of tuples that are sorted, each $\{\alpha_{2,\ell}\}$ can be written sequentially on the second tape one after the other for $\ell \in [n_{\mathsf{PPE}}]$ while the machine makes a pass over the sorted input α_1. Thus the turing machine takes $O(n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}} \cdot \mathsf{poly}(\log(n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}} \cdot p))))$ time for some polynomial poly, and therefore by Lemma 4.2.5, it can be converted to a circuit of size $O(n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}} \cdot \mathsf{poly}(\log(n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}} \cdot p))))$ for some polynomial poly.

For every $r \in [m_{\mathsf{PPE}}]$, we now describe:

Circuit $G_{3,r}$: The output of the circuit $G_{3,r}$ consists of $(\alpha_{3,r}, \mathsf{flag}_r \in \{0,1\})$. The circuit $G_{3,r}$ is described as follows. Let Q_r be the r^{th} at most degree d monomial in $\mathcal{Q}$. The input taken by $G_{3,r}$ is $\{\alpha_{2,\ell}\}_{\ell \in [Q_r]}$. Then the circuit does the following:

1. Combine the tuples $\{\alpha_{2,\ell}\}_{\ell \in Q_r}$ and let their union be A_r. Let the tuples be of the form $(i, j, (j_1, (j_2, j_3))), e_{j,i}, x_{j,i})$ where $e_{j,i} \neq 0$ and $i \in [Q_r]$. The tuples are sorted in the ascending order with respect to the component j_1, and then with respect to j_2, and then with respect to j_3, and finally with respect to i. Let this rearranged set of tuples be B_r

2. After the step above, we get B_r where the tuples are arranged with respect to $j_1 \in [t_1]$ first, and even with fixed j_1 they are sorted with respect to $(j_2, j_3) \in [T] \times [T]$ and even within those fixed, with respect to $i \in Q_r$. Then, make a pass over the tuples in B_r.

Set $\mathsf{flag}_r = 0$ if upon doing a pass on the tuples, we encounter some j_1, for which the number of (j_2, j_3) for which there are tuples in B_r exceed t_2. Otherwise set $\mathsf{flag}_r = 1$.

3. Compute $\alpha_{3,r}$ which consists of a preprocessing of B_r, done as follows. If $\mathsf{flag}_r = 0$, set $\alpha_{3,r} = \perp$, otherwise, take a pass over the tuples in B_r, and in doing so, if we encounter upto d tuples with same value of j (and hence same value of $j_1, (j_2, j_3)$, but potentially different values of i, $x_{j,i}$ and $e_{j,i}$) replace them with a single tuple $(Q_r, j, (j_1, (j_2, j_3)))$. Further these tuples are sorted with respect to j_1 and within the ones with same j_1, they are sorted with respect to j_2 and j_3.

At the end of this step, it outputs $(\mathsf{flag}_r, \alpha_{3,r})$. If $\mathsf{flag}_r = 0$, $\alpha_{3,r} = \perp$. Otherwise if $\mathsf{flag}_r = 1$ (which happens with overwhelming probability), then $\alpha_{3,r}$ consists of upto a $O(k_{\mathsf{PPE}}^{1-\delta})$ sized list of the form $(Q_r, j, (j_1, (j_2, j_3)))$. These corresponds to the corrections that need to be done for the monomial Q_r. In the next step, we will show how to use this information to compute $(\mathsf{S}_1, \ldots, \mathsf{S}_{m_{\mathsf{PPE}}})$ along with flag. We now argue the run time of this circuit.

<u>Size of $G_{3,r}$:</u> Note that the first step can be done by a sorting network and hence by Lemma 4.2.4 it can be implemented by a circuit in size $O((k_{\mathsf{PPE}}^{1-\delta} + t_2 k_{\mathsf{PPE}}^{\frac{1-\delta}{2}}) \cdot \mathsf{poly}(\log_2(k_{\mathsf{PPE}} \cdot n_{\mathsf{PPE}} \cdot p))) = O(k_{\mathsf{PPE}}^{1-\delta}\,\mathsf{poly}(\log_2(n_{\mathsf{PPE}} k_{\mathsf{PPE}} p)))$.

The second step can be done by a multi-tape turing machine with a constant number of tapes which keeps the input B_r on one of the tapes and makes a single pass on that input. It uses the other tape for computing the flag_r. Since the tuples are sorted, only a single pass suffices. This can therefore be computed by a circuit by Lemma 4.2.5 with size $O(k_{\mathsf{PPE}}^{1-\delta}\,\mathsf{poly}(\log_2(n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}} \cdot p)))$.

Finally, the third step can also be done a multi-tape turing machine with a constant number of tapes that keeps B_r on one of its tape, and then makes a pass over that input. It pre-processes the tuples and write it on the other tape in the format described above, while making sure that the written tuple is not duplicated. Since B_r is always sorted, a single pass on the input suffices. This can therefore be computed by a circuit by Lemma 4.2.5 with size

$O(k_{\mathsf{PPE}}^{1-\delta}\,\mathsf{poly}(\log_2(n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}} \cdot p)))$.

Circuit G_4: This circuit takes as input the following:

- $\{\mathsf{flag}_r\}_{r\in[m_{\mathsf{PPE}}]}$ along with flag_{G_2}.
- $\{e_{j,i}, x_{j,i}\}_{j\in[k_{\mathsf{PPE}}],i\in[n_{\mathsf{PPE}}]}$.
- $\{\alpha_{3,r}\}_{r\in[m_{\mathsf{PPE}}]}$.

It proceeds by doing the following steps:

- Compute $\mathsf{flag} = \mathsf{flag}_{G_2}\prod_{r\in[m_{\mathsf{PPE}}]}\mathsf{flag}_r$. This is one of the outputs.
- Make a pass on the combined input $\{\alpha_{3,r}\}_{r\in[m_{\mathsf{PPE}}]}$. For every tuple $(Q_r, j, (j_1, (j_2, j_3)))$ replace it with $(Q_r, j, (j_1, (j_2, j_3)), \{e_{j,i}, x_{j,i}\}_{i\in Q_r})$. This can be done using a RAM lookup program with input database $\{e_{j,i}, x_{j,i}\}_{j\in[k_{\mathsf{PPE}}],i\in[n_{\mathsf{PPE}}]}$. Let the updated output containing these tuples be $\{\beta_r\}_{r\in[m_{\mathsf{PPE}}]}$.
- This is the output generating step. For $r \in [m_{\mathsf{PPE}}]$, one by one, read flag_r in parallel with β_r.
  1. If $\mathsf{flag}_r = 0$, output $\mathsf{S}_r = \{\mathbf{U}_{r,\gamma}, \mathbf{V}_{r,\gamma}^\top\}_{\gamma\in[t_1]}$ where $\mathbf{U}_{r,\gamma} = \mathbf{V}_{r,\gamma}^\top = 0^{T\times t_2}$.
  2. If $\mathsf{flag}_r = 1$, do the following. First let $\beta_r = \{\beta_{r,\gamma}\}_{\gamma\in[t_1]}$ where $\beta_{r,\gamma}$ consists of all tuples in β_r where $j_1 = \gamma$ (i.e. of the form $(Q_r, j, j_1 = \gamma, (j_2, j_3), \{e_{j,i}, x_{j,i}\}_{i\in Q_r})$). Further, the circuit $G_{3,r}$ ensures that these tuples are sorted with respect to (j_2, j_3). Let $D_{r,\gamma}$ denote the number of tuples in $\beta_{r,\gamma}$. Note that $D_{r,\gamma} \le t_2$ (as ensured by setting $\mathsf{flag}_r = 1$). For every $\gamma \in [t_1]$:
     (a) For $l \in [t_2]$, set vectors $\mathbf{u}_{r,\gamma,l} = \mathbf{v}_{r,\gamma,l} = 0^T$.
     (b) For $l \in D_{r,\gamma}$, parse l^{th} tuple in $\beta_{r,\gamma}$ as $(Q_r, j, \gamma, (j_2, j_3), \{e_{j,i}, x_{j,i}\}_{i\in Q_r})$. Set $\mathbf{u}_{r,\gamma,l}[j_2] = \Pi_{i\in Q_r} x_{j,i} - \Pi_{i\in Q_r}(x_{j,i}+e_{j,i})$ (which is equal to $\mathsf{Corr}_{r,j}$) and $\mathbf{v}_{r,\gamma,l}[j_3] = 1$.

(c) Set $\mathbf{U}_{r,\gamma}$ as concatenation of vectors $\{\mathbf{u}_{r,\gamma,l}\}_{l\in[t_2]}$ and $\mathbf{V}_{r,\gamma}^{\top}$ as concatenation of $\{\mathbf{v}_{r,\gamma,l}\}_{l\in[t_2]}$. Output $\mathsf{S}_r = \{\mathbf{U}_{r,\gamma}, \mathbf{V}_{r,\gamma}^{\top}\}_{\gamma\in[t_1]}$.

We first argue about the size of the circuit implementing G_4 and then argue its correctness. Size of G_4: The step of computing flag can be implemented by circuit that has a size of $O(m_{\mathsf{PPE}})$ gates. The second circuit makes a pass on the input consisting of $\{\alpha_{3,r}\}_{r\in[m_{\mathsf{PPE}}]}$, each consisting of atmost $O(k_{\mathsf{PPE}}^{1-\delta})$ tuples. Then it makes at most $q = O(m_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}^{1-\delta})$ lookups of field elements from the database consisting of $n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}$ field elements. This can be implemented by a circuit that runs in size $(n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}} + m_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}^{1-\delta})\, \mathsf{poly}(\log_2(p \cdot n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}))$ for some polynomial poly due to Lemma 4.2.6. The third step can be simulated by a multi-tape turing machine with a constant number of tapes in $O(m_{\mathsf{PPE}} \cdot t_1 \cdot T \cdot t_2\, \mathsf{poly}(\log_2 n_{\mathsf{PPE}} \cdot p \cdot k_{\mathsf{PPE}}))$ steps. And thus, by Lemma 4.2.5 it can be implemented by a circuit of size $O(m_{\mathsf{PPE}} \cdot t_1 \cdot T \cdot t_2\, \mathsf{poly}(\log_2 n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}} \cdot p))$ for some polynomial poly. Let us elucidate: For every $r \in [m_{\mathsf{PPE}}]$, the Turing machine keeps β_r on one of its tape. It keeps flag_r on another tape. The heads on both these tapes move in forward direction"processing" sequentially.

- For any $r \in [m_{\mathsf{PPE}}]$, if $\mathsf{flag}_r = 0$ it writes $2 \cdot t_1 \cdot t_2 \cdot T$ field elements (which consists S_r which consists of all 0 matrices) on a third output tape, which takes $O(t_1 \cdot t_2 \cdot T\, \mathsf{poly}(\log_2 n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}} \cdot p))$ steps.

- In the condition where $\mathsf{flag}_r = 1$, the circuit parses the tuples in β_r sequentially. There are at most $O(k_{\mathsf{PPE}}^{1-\delta}) = O(t_1)$ tuples in all that are divided into $\{\beta_{r,\gamma}\}_{\gamma\in t_1}$. These corresponds to tuples with $j_1 = \gamma$ and they are stored in a sorted fashion with respect to j_1. Thus, the machine makes a pass for every $\gamma \in [t_1]$ and process an output $\mathbf{U}_{r,\gamma}, \mathbf{V}_{r,\gamma}$. This is done by sequentially processing tuples in $\beta_{r,\gamma}$. For every tuple encountered (and say it is the l^{th} tuple where $l \leq D_{r,\gamma} \leq t_2$) of the form $(Q_r, j, \gamma, (j_2, j_3), \{e_{j,i}, x_{j,i}\}_{i\in Q_r})$, it writes two vector $\mathbf{u}_{r,\gamma,l}$ and $\mathbf{v}_{r,\gamma,l} \in \mathbb{Z}_p^T$ where $\mathbf{u}_{r,\gamma,l}[j_2] = \mathsf{Mon}_{Q_r}(\mathbf{x}_j) - \mathsf{Mon}_{Q_r}(\mathbf{x}_j + \mathbf{e}_j)$ and 0 otherwise. Likewise, $\mathbf{v}_{r,\gamma,l}[j_3] = 1$ and 0 otherwise. If the number of tuples is less than t_2, it prints $t_2 - D_{r,\gamma}$ additional zero vectors $\{\mathbf{u}_{r,\gamma,l}, \mathbf{v}_{r,\gamma,l}\}_{l\in[t_2]\setminus[D_{r,\gamma}]}$. This can be done by making a single forward pass on $\beta_{r,\gamma}$ and doing $O(T)$ arithmetic/comparision operations on it per tuple. Thus, it takes at most $O(t_2 \cdot T\,\mathsf{poly}(\log_2(n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}} \cdot p)))$ steps to output $\{\mathbf{U}_{r,\gamma}, \mathbf{V}_{r,\gamma}^\top\}$. Thus the total time complexity to output S_r is $O(t_1 \cdot t_2 \cdot T\,\mathsf{poly}(\log_2(p \cdot k_{\mathsf{PPE}} \cdot n_{\mathsf{PPE}})))$ for some polynomial poly. Thus, the complexity to output $(\mathsf{S}_1, \ldots, \mathsf{S}_{m_{\mathsf{PPE}}})$ is $O(m_{\mathsf{PPE}} \cdot t_1 \cdot t_2 \cdot T\,\mathsf{poly}(\log_2(n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}} \cdot p)))$ for some polynomial poly.

We now argue the correctness of the circuit G_4. $G_{3,r}$ outputs an indicator flag_r which is 1 (with overwhelming probability). If this flag is set to one, $G_{3,r}$ outputs $\alpha_{3,r}$ which consists of all the indices $j \in [k_{\mathsf{PPE}}]$ for which $\mathsf{Mon}_{Q_r}(\mathbf{x}_j) \neq \mathsf{Mon}_{Q_r}(\mathbf{x}_j + \mathbf{e}_j)$. This is the list of indices which need correction for monomial Q_r. Then, for every tuple that $\alpha_{3,r}$ contains, we load up all the inputs and the errors that it needs to do this correction. This updated input is β_r. With overwhelming probability (that is when $\mathsf{flag}_r = 1$) the size of $\beta_{r,\gamma}$ for every $\gamma \in [t_1]$ is less than or equal to t_2. Then, all these corrections are embedded inside the matrices $\{\mathbf{U}_{r,\gamma}, \mathbf{V}_{r,\gamma}^\top\}$ where every correction is embedded in one of the t_2 distinct T dimensional columns of $\mathbf{U}_{r,\gamma}$ and $\mathbf{V}_{r,\gamma}^\top$. Further, point *b*) above ensures that for every $j \in [k_{\mathsf{PPE}}]$ with $\phi(j) = (j_1, (j_2, j_3))$ if the monomial $\mathsf{Mon}_{Q_r}(\mathbf{x}_j)$ was the l^*th monomial for the set Q_r corrected in the j_1th bucket then,

$$\begin{aligned}
\mathsf{Corr}_{r,j} =& \mathbf{u}_{r,j_1,l^*} \cdot \mathbf{v}_{r,j_1,l^*}^\top[j_2, j_3] && ,\\
=& \sum_{l \in [t_2]} \mathbf{u}_{r,j_1,l} \cdot \mathbf{v}_{r,j_1,l}^\top[j_2, j_3] \quad \text{(single monomial corrected in each } |D_{r,j_1}| \leq t_2 \text{ iterations)}, &&\\
=& \mathbf{U}_{r,j_1} \cdot \mathbf{V}_{r,j_1}[j_2, j_3] && .
\end{aligned}$$

Because of this G_4 produces the right output.

Overall circuit-size calculation: Hiding the polynomial in logarithmic multiplicative factors in $n_{\mathsf{PPE}}, p, k_{\mathsf{PPE}}$ withing $\tilde{O}(\cdot)$ we get the following:

- $\mathsf{size}_1 = \tilde{O}(n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}^2)$,
- $\mathsf{size}_2 = \tilde{O}(k_{\mathsf{PPE}}^{\lceil \frac{d}{2} \rceil})$,

- $\mathsf{size}_{G_1} = \tilde{O}(n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}})$
- $\mathsf{size}_{G_2} = \tilde{O}(n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}})$
- $\mathsf{size}_{G_3} = \tilde{O}(m_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}^{1-\delta})$ (adding the size of all m_{PPE} sub-circuits)
- $\mathsf{size}_{G_4} = \tilde{O}(m_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}^{1-\frac{2\delta}{5}})$.

Combining these observations, we get our result. $\square$

Summing up: From the above theorems, we have the following result:

Theorem 4.2.2. *Assuming δ-*LPN *assumption (Definition 4.1.1) holds for any constant $\delta > 0$, then there exists a* PPE *scheme satisfying Definition 3.1.3. Further, if the assumption is subexponentially secure, then so is the resulting* PPE *scheme.*

CHAPTER 5

Amortized Randomized Encoding

In this chapter we give an overview of how to construct an ARE scheme (see Definition 3.1.8). The construction is actually a straightforward adaptation of Yao's Garbled circuits [Yao86], to use PRG in NC^0 to compute garbled tables (rather than a secret-key encryption scheme).

5.1 Amortized Randomized Encoding

Recall again the task in an ARE scheme (See Definition 3.1.8 for formal details). The scheme cares about a function class $\mathcal{F}_{\mathsf{ARE},n_{\mathsf{ARE}},m_{\mathsf{ARE}},k_{\mathsf{ARE}},\lambda}$ which consists of all boolean circuits that take $n_{\mathsf{ARE}}(\lambda)$ number of inputs, produce $m_{\mathsf{ARE}}(\lambda) \cdot k_{\mathsf{ARE}}(\lambda)$ number of outputs, where each output bit is computed by a circuit of size λ. An ARE scheme is just a randomized encoding scheme in NC^0 that is tailor-made to be compatible with a PPE scheme. Namely,

- There exists a constant degree d monomial pattern $\mathcal{Q}$ of size m'_{ARE} over n'_{ARE} variables where $m'_{\mathsf{ARE}} = \tilde{O}(n_{\mathsf{ARE}} + m_{\mathsf{ARE}})$ and $n'_{\mathsf{ARE}} = \tilde{O}(n_{\mathsf{ARE}} + m_{\mathsf{ARE}}^{1-\epsilon})$ for some constant $\epsilon > 0$,
- For any circuit C in the function class, $\mathsf{Encode}(C, \cdot)$ takes as input $\mathbf{x}$ and randomness $\mathbf{r}$. The randomness can be parsed into k_{ARE} equal sized partitions $\mathbf{x}_1, \ldots, \mathbf{x}_{k_{\mathsf{ARE}}}$. Define $\mathbf{a}_i := (\mathbf{x}, \mathbf{r}_i)$. Then, for any bit j, $\mathsf{Encode}(C, \cdot)|_j$ can be computed by an efficiently generatable polynomial of the form:

$$\mathsf{Encode}(C, \cdot)|_j = \sum_{Q \in \mathcal{Q}, i \in [k_{\mathsf{ARE}}]} \mu_{Q,i} \mathsf{Mon}_Q(\mathbf{a}_i),$$

where $\mu_{Q,i} \in \mathbb{Z}$. In other words, the polynomial computing every bit j is in the function class $\mathcal{F}_{\mathsf{PPE},d,n'_{\mathsf{ARE}},\mathcal{Q},k_{\mathsf{ARE}}}$.

The first condition of requiring n'_{ARE} to be sublinear in m_{ARE}, and $m'_{\mathsf{ARE}} = \tilde{O}(m_{\mathsf{ARE}})$ is to ensure that that when we compose ARE with PPE to construct PRE, the PRE satisfies sublinearity property. The second condition is to ensure that ARE can actually be composed with the PPE scheme. We now discuss our approach to build an ARE scheme.

5.1.1 Overall Approach

Let $C^1, \dots C^{m_{\mathsf{ARE}} \cdot k_{\mathsf{ARE}}}$ be the circuits of size λ computing individual output bits of the circuit C. Let C_i for $i \in [k_{\mathsf{ARE}}]$ denote the circuit computing the i^{th} block of m_{ARE} output bits. Namely $C_i = (C^{(i-1)k_{\mathsf{ARE}}+1}, \dots, C^{(i-1)k_{\mathsf{ARE}}+m_{\mathsf{ARE}}})$. Our first observation is that decomposability property of Yao's Garbled Circuits/Randomized Encoding [Yao86] could be useful.

First Idea. Our first idea to set:

$\mathsf{ARE.Encode}(C, \mathbf{x}, \mathbf{r}_1, \dots, \mathbf{r}_{k_{\mathsf{ARE}}}) = (\mathsf{YaoGb.Encode}(C_1, \mathbf{x}; \mathbf{r}_1), \dots, \mathsf{YaoGb.Encode}(C_{k_{\mathsf{ARE}}}, \mathbf{x}; \mathbf{r}_{k_{\mathsf{ARE}}}))$.

Where $\mathsf{YaoGb.Encode}(C_i, \mathbf{x}, \mathbf{r}_i)$ is Yao's garbling of circuit C_i, with input $\mathbf{x}$, computed using randomness $\mathbf{r}_i$. In order to execute this plan we will need to ensure the following things:

- Size of each randomness $\mathbf{r}_i$ is sublinear in the size of C_i (i.e. $O(n'_{\mathsf{ARE}}) = O((n_{\mathsf{ARE}} + m_{\mathsf{ARE}}^{1-\epsilon})\,\mathsf{poly}(\lambda))$ for some constant $\epsilon > 0$),
- Each garbling can be computed by an NC^0 circuit in $(\mathbf{x}, \mathbf{r}_i)$,
- There exists a constant degree d monomial pattern $\mathcal{Q}$ of size $m'_{\mathsf{ARE}} = O((n_{\mathsf{ARE}} + m_{\mathsf{ARE}})\,\mathsf{poly}(\lambda))$ (independent of C_i) such that $\mathsf{YaoGb.Encode}(C_i, \mathbf{x}, \mathbf{r}_i)$ is computable by linear combinations of only those monomials in $\mathbf{x}, \mathbf{r}_i$.

In order to ensure all these three properties, let us examine one of the k_{ARE} components of the candidate ARE.Encode algorithm. Without loss of generality, $\mathsf{YaoGb.Encode}(C_1, \mathbf{x}, \mathbf{r}_1)$.

Inspecting Yao: Recall how $\mathsf{YaoGb.Encode}(C_1, \mathbf{x}, \mathbf{r}_1)$ can be computed. We recall the point and permute formulation[BMR90]:

- Expand $\mathbf{r}_1$ to a sufficient length using a PRG to compute strings $(\sigma, \mathbf{b})$. The string σ consists of two λ bit labels $\{\sigma_{w,0}, \sigma_{w,1}\}_w$ for every wire w that occurs in the circuit. Similarly $\mathbf{b}$ consists of bits $b_w \in \{0,1\}$ for every wire that occurs in the circuit. Here PRG is only used to ensure that the length of the randomness is sublinear.

- For every gate g with input wires w_1 and w_2 and output wire w_3, generate the garbled gate T_g which has four entries:

$$\begin{aligned}
&\mathsf{Enc}(\sigma_{w_1,b_{w_1}},\ \mathsf{Enc}(\sigma_{w_2,b_{w_2}}, (\sigma_{w_3,g(b_{w_1},b_{w_2})})||g(b_{w_1}, b_{w_2}) \oplus b_{w_3}))),\\
&\mathsf{Enc}(\sigma_{w_1,b_{w_1}},\ \mathsf{Enc}(\sigma_{w_2,\bar{b}_{w_2}}, (\sigma_{w_3,g(b_{w_1},\bar{b}_{w_2})})||g(b_{w_1}, \bar{b}_{w_2}) \oplus b_{w_3}))),\\
&\mathsf{Enc}(\sigma_{w_1,\bar{b}_{w_1}},\ \mathsf{Enc}(\sigma_{w_2,b_{w_2}}, (\sigma_{w_3,g(\bar{b}_{w_1},b_{w_2})})||g(\bar{b}_{w_1}, b_{w_2}) \oplus b_{w_3}))),\\
&\mathsf{Enc}(\sigma_{w_1,\bar{b}_{w_1}},\ \mathsf{Enc}(\sigma_{w_2,\bar{b}_{w_2}}, (\sigma_{w_3,g(\bar{b}_{w_1},\bar{b}_{w_2})})||g(\bar{b}_{w_1}, \bar{b}_{w_2}) \oplus b_{w_3}))).
\end{aligned}$$

  where $\bar{b} = 1 - b$ for any bit $b \in \{0,1\}$.

- Let $w_{inp,1}, \ldots, w_{inp,n_{\mathsf{ARE}}}$ denote the input wires. Let $w_{out,1} \ldots, w_{out,m_{\mathsf{ARE}}}$ denote the output wires, and $\{w_g\}_{g \in \mathsf{gate}(C_1)}$ where $\mathsf{gate}(\cdot)$ denote the set of gates function. The output of garbling consists of:

  1. Labels for input wires: $\{(\sigma_{w_{inp,i},x_i}||b_{w_{inp,i}} \oplus x_i)\}_{i \in [n_{\mathsf{ARE}}]}$,
  2. Garbled gate tables for each gate $\{\mathsf{T}_g\}_{g \in \mathsf{gate}(C_1)}$, and,
  3. Output gate translation table $\{(a, \sigma_{w_{out,i},a})\}_{a \in \{0,1\}, i \in [m_{\mathsf{ARE}}]}$.

The evaluation procedure is identical to the one described in [BMR90]. The number of wires in all are bounded by $\tilde{O}(n_{\mathsf{ARE}} + m_{\mathsf{ARE}})$. The total output length is therefore $\tilde{O}(n_{\mathsf{ARE}} + m_{\mathsf{ARE}})$ where $\tilde{O}$ hides polynomial factors in λ. We identify one issue at a time and address them one by one. Once all the issues are addressed the resulting scheme is an ARE scheme.

Using PRG in NC^0. One of the main issue with the scheme above is that the computation above is not in NC^0. Namely, $\sigma, \mathbf{b}$ as well as the double encryptions are not computed by an NC^0 circuit.

We first replace PRG with a PRG in NC^0 with polynomial stretch. Note that length of $\sigma, \mathbf{b}$ is $\tilde{O}(n_{\mathsf{ARE}} + m_{\mathsf{ARE}})$. Since the PRG has polynomial stretch, in order to generate $\sigma, \mathbf{b}$ it suffices to have $\mathbf{r}_i$ of the length sublinear in $n_{\mathsf{ARE}} + m_{\mathsf{ARE}}$ (that is, $\tilde{O}((n_{\mathsf{ARE}} + m_{\mathsf{ARE}})^{1-\epsilon})$ for some constant $\epsilon > 0$). The second issue is to show that all the components of garbled circuit can be computed by an NC^0 circuit in $\mathbf{x}, \sigma, \mathbf{b}$. We make the following observations.

- Labels for input wires can be computed as $\{x_i \cdot \sigma_{w_{inp,i},1} + (1 - x_i)\sigma_{w_{inp,i},0} || x_i \oplus b_{w_{inp,i}}\}_{i \in [n_{\mathsf{ARE}}]}$ which is in NC^0.

- Output gate translation table is already in NC^0.

The missing piece is therefore to ensure that the garbled gate tables are also computable in NC^0. Our solution is to use a PRG in NC^0 with constant stretch to generate double encryptions via one-time pads. To make this formal, let $\mathsf{H} : \{0,1\}^\lambda \to \{0,1\}^{2\cdot\lambda+2}$ denote a PRG in NC^0. Let H_0 denote the function that computes the first half of the output and H_1 denote the function that computes the second half of the output. The garbled gate can be constructed as follows:

$$T_{\mathsf{gate}} = \begin{pmatrix} \mathsf{H}_0(\sigma_{w_1,b_{w_1}}) \oplus \mathsf{H}_0(\sigma_{w_2,b_{w_2}}) \oplus \left(\sigma_{w_3,g(b_{w_1},b_{w_2})} || g(b_{w_1}, b_{w_2}) \oplus b_{w_3}\right) \\ \mathsf{H}_1(\sigma_{w_1,b_{w_1}}) \oplus \mathsf{H}_0(\sigma_{w_2,\bar{b}_{w_2}}) \oplus \left(\sigma_{w_3,g(b_{w_1},\bar{b}_{w_2})} || g(b_{w_1}, \bar{b}_{w_2}) \oplus b_{w_3}\right) \\ \mathsf{H}_0(\sigma_{w_1,\bar{b}_{w_1}}) \oplus \mathsf{H}_1(\sigma_{w_2,b_{w_2}}) \oplus \left(\sigma_{w_3,g(\bar{b}_{w_1},b_{w_2})} || g(\bar{b}_{w_1}, b_{w_2}) \oplus b_{w_3}\right) \\ \mathsf{H}_1(\sigma_{w_1,\bar{b}_{w_1}}) \oplus \mathsf{H}_1(\sigma_{w_2,\bar{b}_{w_2}}) \oplus \left(\sigma_{w_3,g(\bar{b}_{w_1},\bar{b}_{w_2})} || g(\bar{b}_{w_1}, \bar{b}_{w_2}) \oplus b_{w_3}\right) \end{pmatrix} \tag{5.1}$$

Note that each entry can be computed by an NC^0 circuit as $\mathsf{H}_0, \mathsf{H}_1$ are NC^0 circuits, $\sigma_{w,x} = x\sigma_{w,1} + (1 - x)\sigma_{w,0}$ for any bit x and any wire w and $g(b_1, b_2)$ is also in NC^0.

Fixing Monomial Pattern Issue. Let us first identify how many monomials are used in the computation of the $\mathsf{YaoGb.Encode}(C_i, \mathbf{x}, \mathbf{r}_i)$. Note that the computation is in NC^0 and the length of the garbling is $\tilde{O}(n_{\mathsf{ARE}} + m_{\mathsf{ARE}})$. Therefore the total number of monomials is also $\tilde{O}(n_{\mathsf{ARE}} + m_{\mathsf{ARE}})$. Let this monomial set be $\mathcal{Q}_i$. However the issue is that $\mathcal{Q}_i$ depends on circuit C_1. In order to fix this issue, instead of garbling directly, we garble the computation for $U(C_i, \mathbf{x})$ where U is the universal circuit.

$$\mathsf{ARE.Encode}(C, \mathbf{x}, \mathbf{r}_1, \ldots, \mathbf{r}_{k_{\mathsf{ARE}}}) = (\mathsf{YaoGb.Encode}(U, (C_1, \mathbf{x}); \mathbf{r}_1), \ldots, \mathsf{YaoGb.Encode}(U, (C_{k_{\mathsf{ARE}}}, \mathbf{x}); \mathbf{r}_{k_{\mathsf{ARE}}})).$$

Note that the universal circuit U is of size $\tilde{O}(|C_i| + |\mathbf{x}|) = \tilde{O}(m_{\mathsf{ARE}} + n_{\mathsf{ARE}})$ and has $\tilde{O}(m_{\mathsf{ARE}} + n_{\mathsf{ARE}})$ input bits.

Thus for executing the garbling, we will need $n'_{\mathsf{ARE}} = \tilde{O}(n_{\mathsf{ARE}} + (m_{\mathsf{ARE}} + n_{\mathsf{ARE}})^{1-\epsilon})$ for some constant $\epsilon > 0$. The number of monomials required is also bounded by $\tilde{O}((m_{\mathsf{ARE}} + n_{\mathsf{ARE}}))$. We now argue that the monomial pattern is independent of the circuit C_i.

The garbling consists of three components: input labels, garbled tables and the translation table. First, it is easy to observe that the garbled tables and output translation table is independent of the circuit C_i as they only depend only on the description of universal circuit. Thus, the monomials involved in those computations are independent of the circuit. We now show that the input wire labels for circuit C_1 can be computed as follows. Let the wires in the universal circuit for the circuit (say C_1) be denoted as $w_{ckt,i}$ for $i \in [\ell_{C_1}]$, where ℓ_{C_1} is the size of C_1. These labels have the following structure:

$$\{C_{1,i}\sigma_{w_{ckt,i},1} + (1 - C_{1,i})\sigma_{w_{ckt,i},0} || C_{1,i} \oplus b_{w_{ckt,i}}\}_{i \in [\ell_{C_1}]}$$

The number of monomials for computing these labels do not depend on the circuit as they can be computed if one can compute $\sigma_{w_{ckt,i},1}$, $\sigma_{w_{ckt,i},0}$ and $b_{w_{ckt,i}}$. The labels for the inputs also do not depend on the circuit as the labels are given by: $\{x_i \cdot \sigma_{w_{inp,i},1} + (1 - x_i)\sigma_{w_{inp,i},0} || x_i \oplus b_{w_{inp,i}}\}_{i \in [n_{\mathsf{ARE}}]}$.

Security The security of the scheme follows immediately from the security of the garbled circuit construction [Yao86, BMR90]. Our construction is an instantiation of the construction of garbling scheme where we have the following:

- We use PRG in NC^0 instead of using any PRG, to bring down the complexity of generating labels and permutation bits in NC^0,
- The double encryptions occurring in the garbled circuit table replaced by one time pad that are generated using PRG in NC^0, and,
- We garble universal computation instead of the computation directly,

As a result, the security directly follows from the security of the garbling scheme and the security of PRG in NC^0.

5.2 Construction Details

Before we proceed further, we give the formal definition of a PRG in NC^0.

Definition 5.2.1. *(Pseudorandom Generator.) A stretch-$m(\cdot)$ pseudorandom generator is a Boolean function $\mathsf{PRG} : \{0,1\}^* \to \{0,1\}^*$ mapping n-bit inputs to $m(n)$-bit outputs (also known as the stretch) that is computable by a uniform p.p.t. machine, and for any non-uniform p.p.t adversary $\mathcal{A}$ there exist a negligible function negl such that, for all $n \in \mathbb{N}$*

$$\left| \Pr_{r \leftarrow \{0,1\}^n} [\mathcal{A}(\mathsf{PRG}(r)) = 1] - \Pr_{z \leftarrow \{0,1\}^m} [\mathcal{A}(z) = 1] \right| < \mathsf{negl}(n).$$

Further, a PRG is said to be in NC^0 if PRG is implementable by a uniformly efficiently generatable NC^0 circuit. PRG is said to have polynomial stretch if $m(n) = n^{1+\Omega(1)}$. Finally, PRG is said to be subexponentially secure if $\mathsf{negl}(n) = O(\exp(-n^{\Omega(1)}))$.

Remark 5.2.1. In the candidate constructions, typically there is a sampling algorithm that samples the description of PRG, and this property of computational indistinguishability is expected to hold with probability $1 - o(1)$ over the choice of PRG. Such a PRG will give us an existential result. Constructively, this issue can be addressed by constructing our FE scheme with multiple instantiations of PRG so that with overwhelming probability, at least one of the FE schemes we build is secure, and then using an FE combiner [ABJ+19, JMS20].

In Figure 5.1, we now give the formal construction of the ARE scheme. We establish some useful notations and recall the tools we need.

Notation: $\lambda \in \mathbb{N}$ is the security parameter, $n_{\mathsf{ARE}}, m_{\mathsf{ARE}}, k_{\mathsf{ARE}}$ are parameters associated with the function class $\mathcal{F}_{\mathsf{ARE},n_{\mathsf{ARE}},m_{\mathsf{ARE}},k_{\mathsf{ARE}}}$. We set $n'_{\mathsf{ARE}} = (n_{\mathsf{ARE}} + m^{1-\epsilon}_{\mathsf{ARE}})\,\mathsf{poly}(\lambda)$. Let $U = U_{m_{\mathsf{ARE}}\lambda,n_{\mathsf{ARE}},m_{\mathsf{ARE}}} : \{0,1\}^{m_{\mathsf{ARE}}\cdot\lambda} \times \{0,1\}^{n_{\mathsf{ARE}}} \to \{0,1\}^{m_{\mathsf{ARE}}}$ be the universal circuit for evaluating circuits with n_{ARE}-bit inputs, m_{ARE}-bit outputs, and size $m_{\mathsf{ARE}} \cdot \lambda$. In particular, $U(C_i, \mathbf{x}) = C_i(\mathbf{x})$ for circuits C_i and input $\mathbf{x}_i$ satisfying the requirements.

Tool: A PRG in NC^0 (denoted by G) that stretches $t^{1-\epsilon}$ bits to t bits. We will set $t = n'_{\mathsf{ARE}} - n_{\mathsf{ARE}}$. A PRG in NC^0 (denoted by H) that stretches λ bits to $2\cdot\lambda + 2$ bits. Denote by H_0 the function that computes first half of the output of H and by H_1 the function that computes the other half.

We now briefly discuss why all the properties are satisfied:

Indistinguishability Security: The security holds readily due to the security of the PRG in NC^0, and security of Yao's garbling scheme [Yao86, BMR90]. Consider two challenge messages $\mathbf{x}_0, \mathbf{x}_1$ with $C(\mathbf{x}_0) = C(\mathbf{x}_1)$. If PRG security holds, then Π computed by encoding $\mathbf{x}_\beta$ for a random $\beta \in \{0,1\}$ is computationally indistinguishable to an honest garbling of the computation $(U(C_1, \mathbf{x}_\beta), \ldots, U(C_{k_{\mathsf{ARE}}}, \mathbf{x}_\beta))$ using truly generated randomness. But due to the security of Yao's garbling scheme, this is indistingusihable to garbling of $(U(C_1, \mathbf{x}_0), \ldots, U(C_{k_{\mathsf{ARE}}}, \mathbf{x}_0))$ which is independent of β.

Efficiency: The efficiency properties have already been argued in the overview. The size of randomness $\mathbf{r}_i$ is $n'_{\mathsf{ARE}} - n_{\mathsf{ARE}} = O((n_{\mathsf{ARE}} + m^{1-\epsilon}_{\mathsf{ARE}})\,\mathsf{poly}(\lambda))$. The computation of $\mathsf{Encode}(\cdot, \mathbf{a}_1, \ldots, \mathbf{a}_{k_{\mathsf{ARE}}})$ is can be computed by polynomials using a d- monomial pattern $\mathcal{Q}$ of size $O((n_{\mathsf{ARE}} + m_{\mathsf{ARE}})\,\mathsf{poly}(\lambda))$ over n'_{ARE} variables for some constant $d > 0$. This is because each Π_κ is computed by an NC^0 circuit on input $\mathbf{a}_\kappa$ of length n'_{ARE}, and has a length of $O((n_{\mathsf{ARE}} + m_{\mathsf{ARE}})\,\mathsf{poly}(\lambda))$. All components of this output are independent of the circuit C_κ,

The ARE scheme

Encode $\mathsf{Encode}(C, \mathbf{x}, \mathbf{r})$**:** Parse $C = (C_1, \ldots, C_{k_{\mathsf{ARE}}})$ such that $C_i : \{0,1\}^{n_{\mathsf{ARE}}} \to \{0,1\}^{m_{\mathsf{ARE}}}$ is the circuit computing the i^{th} chunk of output of C of size m_{ARE}. The size of circuit C_i is $m_{\mathsf{ARE}}\lambda$. Parse $\mathbf{r} = (\mathbf{r}_1, \ldots, \mathbf{r}_{k_{\mathsf{ARE}}})$ where $\mathbf{r}_i \in \{0,1\}^{n'_{\mathsf{ARE}} - n_{\mathsf{ARE}}}$. Set $\mathbf{a}_i = (\mathbf{x}, \mathbf{r}_i) \in \{0,1\}^{n'_{\mathsf{ARE}}}$ for $i \in [k_{\mathsf{ARE}}]$. For every $\kappa \in [k_{\mathsf{ARE}}]$, compute Π_κ as follows:

- Using G expand $\mathbf{r}_\kappa$ into $(\sigma, \mathbf{b})$ of length $(n_{\mathsf{ARE}} + m_{\mathsf{ARE}})\,\mathsf{poly}(\lambda)$. Here σ will be used as labels to produce garbling of $U(C_\kappa, \mathbf{x})$ and $\mathbf{b}$ will be used as permutation bits for every wire in the circuit U. Precisely, for every wire w in U, we let $\sigma_{w,0}, \sigma_{w,1} \in \{0,1\}^\lambda$ be the two labels for the wire, and $b_w \in \{0,1\}$ the permutation bit for the wire.

- (Input wire labels for C_κ and $\mathbf{x}$) Generate input labels of $(C_\kappa, \mathbf{x})$. That is for every input wire $w_{ckt,i}$ for $i \in [m_{\mathsf{ARE}} \cdot \lambda]$ and $w_{inp,j}$ for $j \in [n_{\mathsf{ARE}}]$.

$$\mathsf{Lab}_{C_\kappa,i} = \sigma_{w_{ckt,i},0}(1 - C_{\kappa,i}) + \sigma_{w_{ckt,i},1}(C_{\kappa,i}) \| C_{\kappa,i} \oplus b_{w_{ckt},i},$$
$$\mathsf{Lab}_j = \sigma_{w_{inp,j},0}(1 - x_j) + \sigma_{w_{inp,j},1}(x_j) \| x_j \oplus b_{w_{inp},j}$$

  Above $C_{\kappa,i}$ is i^{th} bit of the circuit description.

- Compute garbled tables for U. That is, for every gate gate in U with input wires w_1, w_2 and output wire w_3, output the following garbled table.

$$T_{\mathsf{gate}} = \begin{pmatrix} \mathsf{H}_0(\sigma_{w_1,b_{w_1}}) \oplus \mathsf{H}_0(\sigma_{w_2,b_{w_2}}) \oplus \left(\sigma_{w_3,g(b_{w_1},b_{w_2})} || g(b_{w_1}, b_{w_2}) \oplus b_{w_3}\right) \\ \mathsf{H}_1(\sigma_{w_1,b_{w_1}}) \oplus \mathsf{H}_0(\sigma_{w_2,\bar{b}_{w_2}}) \oplus \left(\sigma_{w_3,g(b_{w_1},\bar{b}_{w_2})} || g(b_{w_1}, \bar{b}_{w_2}) \oplus b_{w_3}\right) \\ \mathsf{H}_0(\sigma_{w_1,\bar{b}_{w_1}}) \oplus \mathsf{H}_1(\sigma_{w_2,b_{w_2}}) \oplus \left(\sigma_{w_3,g(\bar{b}_{w_1},b_{w_2})} || g(\bar{b}_{w_1}, b_{w_2}) \oplus b_{w_3}\right) \\ \mathsf{H}_1(\sigma_{w_1,\bar{b}_{w_1}}) \oplus \mathsf{H}_1(\sigma_{w_2,\bar{b}_{w_2}}) \oplus \left(\sigma_{w_3,g(\bar{b}_{w_1},\bar{b}_{w_2})} || g(\bar{b}_{w_1}, \bar{b}_{w_2}) \oplus b_{w_3}\right) \end{pmatrix} \tag{5.2}$$

- Let $w_{out,\gamma}$ for $\gamma \in [m_{\mathsf{ARE}}]$ denote the wires for output. Generate output translation table $\mathsf{OutTab} = \{(0, \sigma_{w_{out,\gamma},0}), (1, \sigma_{w_{out,\gamma},1})\}_{\gamma \in [m_{\mathsf{ARE}}]}$. Set $\Pi_\kappa = \{\mathsf{Lab}_{C_{\kappa,i}}, \mathsf{Lab}_j, T_{\mathsf{gate}}, \mathsf{OutTab}\}_{i \in [m_{\mathsf{ARE}} \cdot \lambda],\ j \in [n_{\mathsf{ARE}}],\ \mathsf{gate} \in \mathsf{gate}(U)}$. The output of the encode operation is $\Pi = \{\Pi_\kappa\}_{\kappa \in [k_{\mathsf{ARE}}]}$.

Decode $\mathsf{Decode}(\Pi = (\Pi_1, \ldots, \Pi_{k_{\mathsf{ARE}}}))$**:** Compute and output $\mathbf{y}_\kappa = \mathsf{YaoDecode}(\Pi_\kappa)$ for $\kappa \in [k_{\mathsf{ARE}}]$.

Figure 5.1: ARE Scheme Description

except one i.e. the labels corresponding to the circuit input C_κ. Here too, the labels can be computed as:

$$\mathsf{Lab}_{C_\kappa,i} = \sigma_{w_{ckt,i},0}(1 - C_{\kappa,i}) + \sigma_{w_{ckt,i},1}(C_{\kappa,i}) \| C_{\kappa,i} \oplus b_{w_{ckt},i}$$

which only require the monomials needed to compute $\sigma_{w_{ckt,i},0}, \sigma_{w_{ckt,i},1}$ and $b_{w_{ckt},i}$ which is independent of C_κ. Thus, we have the following theorem:

Theorem 5.2.1. *Assuming the existence of a boolean* PRG *in* NC^0 *with a stretch* $n^{1+\epsilon}$ *for some constant* $\epsilon > 0$ *where* n *is the input length to the* PRG *(see Definition 5.2.1), then there exists an* ARE *scheme satisfying Definition 3.1.8. Further, if the* PRG *is subexponentially secure, then so is* ARE*.*

CHAPTER 6

Partially Hiding Functional Encryption

In this chapter, we discuss the notion of a Partially Hiding Functional Encryption scheme PHFE (see Definition 2.4.1 for the requirements of a PHFE scheme). This is the only primitive in the construction that requires an assumption related to bilinear maps, namely the DLIN assumption. In Section 6.1, we recall some notations as well preliminaries about bilinear maps. In Section 6.2 we give an overview as well as the construction of such a PHFE scheme.

6.1 Notations and Bilinear Map Preliminaries

By $\otimes$ we denote the tensor product (Kronecker product) of two matrices. For any matrices $\mathbf{A} \in \mathbb{Z}_p^{n_1 \times n_2}$ denoted as $[a_{i,j}]_{i \in [n_1], j \in n_2}$ and a matrix $\mathbf{B} \in \mathbb{Z}_p^{n_3 \times n_4}$, $\mathbf{P} = \mathbf{A} \otimes \mathbf{B}$ is a matrix in $\mathbb{Z}_p^{n_1 \cdot n_3 \times n_2 \cdot n_4}$ which looks like the following:

$$\mathbf{P} = \begin{bmatrix} a_{1,1}\mathbf{B} & a_{1,2}\mathbf{B} \ldots & a_{1,n_2}\mathbf{B} \\ \vdots & \ddots & \\ a_{n_1,1}\mathbf{B} & & a_{n_1,n_2}\mathbf{B} \end{bmatrix}.$$

We also make use of the mixed product property of tensor product which says that:

$$(\mathbf{A} \otimes \mathbf{B}) \cdot (\mathbf{C} \otimes \mathbf{D}) = (\mathbf{A} \cdot \mathbf{C} \otimes \mathbf{B} \cdot \mathbf{D}),$$

for any matrices $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, $\mathbf{D}$ with the dimensions compatible with the operations involved.

6.1.1 Prime Order Bilinear Maps

A bilinear map is a tuple $(e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_1, g_T)$ where $\mathbb{G}_b$ for $b \in \{1, 2, T\}$ is a group of order p for some prime, which is generated by g_b. It is equipped with an efficiently computable pairing operation with the following property:

$$e(g_1^x, g_2^y) = g_T^{x \cdot y},$$

for all $x, y \in p$. Therefore, the pairing enables quadratic computations in the exponents. Such bilinear maps have been studied for a long time in cryptography, for a number of applications (for example [GS08, BKKV10, OT10, BJK15, JR13]) and a number of different assumptions have been proposed.

Assumptions over Bilinear Maps. Since $\mathbb{G}_1$ and $\mathbb{G}_2$ are of the same prime order, there will always exist an isomorphism between the two groups. Depending on whether this isomorphism is easy to find, the assumptions can be characterized based on that. For example, if the isomorphism is hard to find, one can plausibly make the DDH assumption over bilinear maps [BGdMM05] which says that for any $b \in \{1, 2\}$:

$$(g_b^x, g_b^y, g_b^{x \cdot y}) \approx_c (g_b^x, g_b^y, g_b^r)$$

where x, y, r are random elements. This assumption is clearly false if the isomorphism is efficienctly computable because one can simply compute g_{3-b}^x from g_b^x and then simply pair it with g_b^y to compute g_T^{xy}.

The main assumption that one makes when there may be an efficiently computable isomorphism between the groups is called DLIN:

$$\begin{gathered}\left\{\left(g^x, g^y, g^{xr}, g^{ys}, g^{r+s}\right) \mid x, y, r, s \leftarrow \mathbb{Z}_p\right\} \\ \approx_c \left\{\left(g^x, g^y, g^{xr}, g^{ys}, g^z\right) \mid x, y, r, s, z \leftarrow \mathbb{Z}_p\right\},\end{gathered}$$

where $g \in \{g_1, g_2\}$. And thus, DLIN is arguably a weaker assumption to make than DDH since it may be expected to hold true, even if there is an efficiently computable isomorphism. We will make this assumption over symmetric bilinear maps where $\mathbb{G}_1 = \mathbb{G}_2$ and $g_1 = g_2$. This assumption was first introduced in the 2004 work of Boneh, Boyen, and Shacham [BBS04]. Since then DLIN and assumptions implied by DLIN have seen extensive use in a wide variety of applications throughout cryptography, such as Identity-Based Encryption, Attribute-Based Encryption, Functional Encryption for degree 2 polynomials, Non-Interactive Zero Knowledge, etc. (See, e.g. [GS08, BKKV10, OT10, BJK15, JR13]). We now formally define the notions.

Definition 6.1.1 (Syntax of a Bilinear Group Generator)**.** *We say that $\mathcal{G}$ is a bilinear group generator if its a polynomial time algorithm that on input the security parameter 1^λ outputs a tuple $(e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, g_T)$ where p is a $\Theta(\lambda)$ bit prime and is also the order of groups $\mathbb{G}_1, \mathbb{G}_2.\mathbb{G}_T$ such that:*

- *The group operation for each of the groups is efficiently computable,*
- *g_1, g_2, g_T are generators for $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ respectively, and,*
- *$e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a non-degenerate, efficiently computable pairing such that $g_T = e(g_1, g_2)$.*

Further, we say that $\mathcal{G}$ is a bilinear group generator for symmetric bilinear groups if $\mathbb{G}_1 = \mathbb{G}_2 = \mathbb{G}$ and $g_1 = g_2$. Note that this can be specified by a shorter tuple of the form $(e, \mathbb{G}, \mathbb{G}_T, p, g, g_T)$.

We now develop some short hand for denoting vectors of group elements. Consider a bilinear map $\mathsf{PP} = (e, \mathbb{G}, \mathbb{G}_T, p, g, g_T)$. Using such a map, we can encode any matrix $\mathbf{M} \in \mathbb{Z}_p^{n_1 \times n_2}$ in the exponent of the group generator g and denote it as $g^{\mathbf{M}}$, which represents as the matrix consisting of component wise exponentiation of g to the entries of $\mathbf{M}$. We define two shorthands: $[\mathbf{M}] := g^{\mathbf{M}}$, and $[\mathbf{M}]_T := g_T^{\mathbf{M}}$. In this notation, for a scalar $x \in \mathbb{Z}_p$, $[x] = g^x$. This notation also gives rise to the following shorthand. For any matrices $\mathbf{A} \in \mathbb{Z}_p^{n_1 \times n_2}$ and

$\mathbf{B} \in \mathbb{Z}_p^{n_2 \times n_3}$ denote by $e([\mathbf{A}], [\mathbf{B}]) = [\mathbf{A} \cdot \mathbf{B}]_T$. Next, we define the k-LIN assumption and the MDDH assumption over the symmetric bilinear maps.

Definition 6.1.2 (k-LIN Assumption)**.** *We say that k-LIN holds with respect to $\mathcal{G}$, if for any polynomial time adversary $\mathcal{A}$, the following advantage is a negligible function:*

$$\begin{aligned}\mathsf{Adv}_{\mathcal{A}}^{\mathsf{k\text{-}LIN}}(\lambda) := \Big| &\Pr[\mathcal{A}(\mathsf{PP}, [x_1], \ldots, [x_k], [x_1 \cdot r_1], \ldots, [x_k \cdot r_k], [r_1 + \ldots + r_k]) = 1] \\ &- \Pr[\mathcal{A}(\mathsf{PP}, \mathsf{PP}, [x_1], \ldots, [x_k], [x_1 \cdot r_1], \ldots, [x_k \cdot r_k], [z]) = 1] \Big|,\end{aligned}$$

where $\mathsf{PP} \leftarrow \mathcal{G}(1^\lambda)$ and $x_1, \ldots, x_k, r_1, \ldots r_k, z \leftarrow \mathbb{Z}_p$.

The DLIN assumption is the same as k-LIN where $k = 2$ and the DDH assumption is the same as k-LIN where $k = 1$. A related assumption is that of MDDH defined below.

Definition 6.1.3 ($\mathsf{MDDH}_{k,\ell}^d$ Assumption)**.** *Let $k, \ell, d \in \mathbb{N}$. We say that $\mathsf{MDDH}_{k,\ell}^d$ holds with respect to $\mathcal{G}$, if for any polynomial time adversary $\mathcal{A}$, the following advantage is a negligible function:*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{MDDH}_{k,\ell}^d}(\lambda) := \Big| \Pr[\mathcal{A}(\mathsf{PP}, [\mathbf{M}], [\mathbf{M} \cdot \mathbf{S}]) = 1] - \Pr[\mathcal{A}(\mathsf{PP}, [\mathbf{M}], [\mathbf{U}]) = 1] \Big|,$$

where $\mathsf{PP} \leftarrow \mathcal{G}(1^\lambda)$, $\mathbf{M} \leftarrow \mathbb{Z}_p^{\ell \times k}$, $\mathbf{S} \leftarrow \mathbb{Z}_p^{k \times d}$ and $\mathbf{U} \leftarrow \mathbb{Z}_p^{\ell \times d}$.

It was shown in [EHK⁺13] that for any $k \geq 2$

$$(k-1)\text{-}\mathsf{LIN} \implies k\text{-}\mathsf{LIN} \implies \mathsf{MDDH}_{k,k+1}^1 \implies \mathsf{MDDH}_{k,\ell}^d,$$

for $\ell > k$, and $d \geq 1$ under a tight security reduction (namely, $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{MDDH}_{k,\ell}^d} = \mathsf{Adv}_{\mathcal{A}'}^{\mathsf{k-LIN}}$). Thus, DLIN is the strongest assumption in this family of k-LIN with $k \geq 2$. Therefore, we will show a construction of a PHFE scheme from k-LIN (equivalently $\mathsf{MDDH}_{k,k+1}^1$) for any $k \geq 2$. The construction from DLIN follows as a corollary.

6.2 Constructing PHFE

The earlier works resulting in the thesis [JLMS19, AJL+19, GJLS21] actually constructed a PHFE scheme from the DDH assumption over assymetric bilinear maps. Over time, this result has been refined and simplified. In this thesis, we recall the state of the art scheme due to Wee [Wee20]. The scheme was actually shown to be secure assuming bilateral k-LIN however, any symmetric group where k-LIN holds trivially is a group that satisfies bilateral k-LIN by setting $\mathbb{G}_1 = \mathbb{G}_2 = \mathbb{G}$.

6.2.1 Overview

First and foremost, we recall briefly the properties that a PHFE scheme must satisfy.

Recalling the requirement. Recall from Definition 2.4.1 that we would like to construct a public key PHFE scheme scheme where an encryptor can encrypt vectors of the form (P, S) where $\mathsf{P}, \mathsf{S} \in \mathbb{Z}_p^{n_{\mathsf{PHFE}}}$ where p is the prime order of the bilinear group PP. The encryption time should be linear in n_{PHFE}. Given the keys, we should be able recover $[f(\mathsf{P}, \mathsf{S})]_T$ where f is a degree $(d, 2)$-polynomial where d is any constant. Further, the scheme should satisfy (selective) single-ciphertext simulation security.

6.2.1.1 Simplified Setting: Constructing Quadratic Functional Encryption

First consider how we can construct a PHFE scheme that supports degree $(0, 2)$-polynomials - functions that are independent of P and are quadratic polynomials in S. As a first step, consider encrypting the message $\mathbf{S}$ using k-LIN as:

$$[\mathbf{v}_1 \cdot \mathbf{A}_1 + \mathsf{S}],\ [\mathbf{v}_2 \cdot \mathbf{A}_2 + \mathsf{S}]$$

where $\mathbf{v}_1, \mathbf{v}_2 \leftarrow \mathbb{Z}_p^{1\times k}$ and matrices $\mathbf{A}_1, \mathbf{A}_2 \leftarrow \mathbb{Z}_p^{k\times n_{\mathsf{PHFE}}}$. These can be computed publicly because we will include $[\mathbf{A}_1],\ [\mathbf{A}_2]$ in the public key.

Let us say that the quadratic function f is of the following form:

$$f(\mathsf{S}) = (\mathsf{S} \otimes \mathsf{S}) \cdot \mathbf{f}^\top$$

for a vector $\mathbf{f} \in \mathbb{Z}_p^{1 \times n_{\mathsf{PHFE}}{}^2}$. Using such a scheme one can compute in the exponent of the target group:

$$\begin{aligned}&(\mathbf{v}_1\mathbf{A}_1 + \mathsf{S}) \otimes (\mathbf{v}_2\mathbf{A}_2 + \mathsf{S})\mathbf{f}^\top \\ =& f(\mathsf{S}) + \text{ cross terms}\end{aligned}$$

The key point is that the cross terms can then be computed as a linear function on $O(k \cdot n_{\mathsf{FE}})$ variables, where the coefficients can be derived from the the function and the public key. Let us analyze the cross terms in the computation of tensor alone:

$$\begin{aligned}(\underbrace{\mathbf{v}_1\mathbf{A}_1 + \mathsf{S}}_{:=\mathbf{z}_1}) \otimes (\underbrace{\mathbf{v}_2\mathbf{A}_2 + \mathsf{S}}_{:=\mathbf{z}_2}) =& \underbrace{(\mathbf{v}_1\mathbf{A}_1 \otimes \mathsf{S} + \mathsf{S} \otimes \mathbf{v}_2\mathbf{A}_2 + \mathbf{v}_1\mathbf{A}_1 \otimes \mathbf{v}_2\mathbf{A}_2)}_{\text{cross terms}} + \mathsf{S} \otimes \mathsf{S} \\ &= \mathbf{v}_1\mathbf{A}_1 \otimes \mathbf{z}_2 + \mathsf{S} \otimes \mathbf{v}_2\mathbf{A}_2 + \mathsf{S} \otimes \mathsf{S} \\ &= (\mathbf{v}_1 \otimes \mathbf{z}_2)(\mathbf{A}_1 \otimes \mathbf{I}_{n_{\mathsf{PHFE}}}) + (\mathsf{S} \otimes \mathbf{v}_2)(\mathbf{I}_{n_{\mathsf{PHFE}}} \otimes \mathbf{A}_2) + \mathsf{S} \otimes \mathsf{S} \\ &= (\mathbf{v}_1 \otimes \mathbf{z}_2)||(\mathsf{S} \otimes \mathbf{v}_2) \cdot \mathbf{M} + \mathsf{S} \otimes \mathsf{S}\end{aligned}$$

Above,

$$\mathbf{M} = \begin{bmatrix} \mathbf{A}_1 \otimes \mathbf{I}_{n_{\mathsf{PHFE}}} \\ \mathbf{I}_{n_{\mathsf{PHFE}}} \otimes \mathbf{A}_2 \end{bmatrix}.$$

A Useful Tool. Thus, in order to implement the idea we use an Public-Key Inner Product FE (IPE) with the following properties.

- The security follows from the k-LIN assumption,
- (Canonical Property) The encryption algorithm takes as input vectors $[\mathbf{x}]$ encoded in the exponent, and the key generation takes as input vectors $[\mathbf{y}]$ encoded in the exponent, as opposed to taking them in the clear. The decryption produces $[\langle \mathbf{x}, \mathbf{y} \rangle]_T$.

- (Selective Single Ciphertext Simulation security). The challenge ciphertext and the function keys can be simulated knowing $\left\{[\mathbf{y}], [\langle \mathbf{x}, \mathbf{y} \rangle]\right\}_{\mathbf{y}}$ for every $\mathbf{y}$ for which keys are issued.

- (Linear Efficiency) The cipher-text and the keys can generated in time $\tilde{O}(k \cdot n_{\mathsf{IPE}})$ where $\tilde{O}$ hides multiplicative factors in the time it takes to compute group operations.

There are many schemes that can be extended easily to ones satisfying these properties [ALS16, ABDP15, Lin17]. One such scheme can be found in [Tom19].

Quadratic FE, summing up. With such a tool in hand, the scheme can be described as follows. The public key consists of:

$$[\mathbf{A}_1], [\mathbf{A}_2], \mathsf{IPE.PK}$$

The master secret key consists of $\mathbf{A}_1, \mathbf{A}_2, \mathsf{IPE.MSK}$. The ciphertext encrypting S, consists of:

$$[\mathbf{v}_1\mathbf{A}_1 + \mathsf{S}], [\mathbf{v}_2\mathbf{A}_2 + \mathsf{S}], \mathsf{IPE.CT} = \mathsf{IPE.Enc}(\mathsf{PK}, [\mathbf{v}_1 \otimes \mathbf{z}_2 || \mathsf{S} \otimes \mathbf{v}_2])$$

The function key for a function f, consists of:

$$\mathsf{IPE.SK}_f = \mathsf{IPE.KeyGen}(\mathsf{PK}, [\mathbf{M}] \cdot \mathbf{f})$$

Finally, in order to decrypt, one can first compute $[(\mathbf{v}_1\mathbf{A}_1 + \mathsf{S}) \otimes (\mathbf{v}_2\mathbf{A}_2 + \mathsf{S})\mathbf{f}]_T$, and then compute $\mathsf{IPE.Dec}(\mathsf{IPE.SK}_f, \mathsf{IPE.CT}) = [(\mathbf{v}_1\mathbf{A}_1 + \mathsf{S}) \otimes (\mathbf{v}_2\mathbf{A}_2 + \mathsf{S})\mathbf{f}^\top - f(\mathsf{S})]_T$. Using this, one can unmask $[f(\mathsf{S})]_T$.

Security. Security also follows readily from the k-LIN assumption and simulation security of IPE (which also follows from k-LIN). Firstly, as such $[\mathbf{z}_1], [\mathbf{z}_2]$ hide information about S due to the security of k-LIN. Secondly, due to the simulation security of IPE scheme, the challenge ciphertexts and all the function keys can be simulated knowing $[\mathbf{M}], [\mathbf{z}_1], [\mathbf{z}_2]$ and $f(\mathsf{S})$ for all queried functions f because the output of $\mathsf{IPE.Dec}$ is $[(\mathbf{v}_1\mathbf{A}_1 + \mathsf{S}) \otimes (\mathbf{v}_2\mathbf{A}_2 + \mathsf{S})\mathbf{f}^\top - f(\mathsf{S})]_T = [(\mathbf{z}_1 \otimes \mathbf{z}_2)\mathbf{f}^\top - f(\mathsf{S})]$

6.2.2 Construction Details for PHFE

The above intuition can be generalized to a PHFE scheme. The only difference is that now the coefficient vector $\mathbf{f}^\top \in \mathbb{Z}_p^{n_{\mathsf{PHFE}}{}^2}$ should be replaced with $(\mathbf{f}(\mathsf{P}))^\top \in \mathbb{Z}_p^{n_{\mathsf{PHFE}}{}^2}$.

In the previous framework, one can still compute $[(\mathbf{v}_1\mathbf{A}_1 + \mathsf{S}) \otimes (\mathbf{v}_2\mathbf{A}_2 + \mathsf{S})(\mathbf{f}(\mathsf{P}))^\top]_T$. However, to recover $[(\mathsf{S} \otimes \mathsf{S})\mathbf{f}(\mathsf{P}))^\top]_T$ from this we need a way to compute $[(\mathbf{v}_1 \otimes \mathbf{z}_2 || \mathsf{S} \otimes \mathbf{v}_2)\mathbf{M}\mathbf{f}(\mathsf{P}))^\top]_T$. In order to do that, the idea is that we replace IPE with its partially hiding analogue. Instead of relying on a canonical IPE scheme, we now rely on a canonical PHFE scheme supporting $(d, 1)$-polynomials. Assuming k-LIN, we can bootstrap any canonical PHFE scheme supporting $(d, 1)$-polynomials to one which supports $(d, 2)$-polynomials, thereby increasing the degree of the polynomial in the secret component by one. We will call this scheme as PHFE_1. The properties that such a scheme should satisfy are:

- The security should follow from the k-LIN assumption,
- (Canonical Property) The encryption should take as input two vectors: A vector P in the clear and a vector $[\mathsf{S}]$ encoded in the exponent. The setup should take as input a matrix $[\mathbf{M}]$ in the exponent. The function f in the function class should have the following structure:

$$f : (\mathsf{P}, \mathsf{S}) \to \mathsf{S} \cdot \mathbf{M}(\mathbf{f}(\mathsf{P}))^\top$$

  where $(\mathbf{f}(\mathsf{P}))^\top$ is a vector, where every component is a degree d polynomial in P. The decryption must compute $[\mathsf{S}\mathbf{M}(\mathbf{f}(\mathsf{P}))^\top]_T$.
- (Selective Single Ciphertext Simulation security). The challenge ciphertext and the function keys should be simulatable knowing $\{[\mathbf{M}], \mathsf{P}, [\mathsf{S}\mathbf{M}(\mathbf{f}_i(\mathsf{P}))^\top], f_i\}_{i \in [q]}$ where f_i's are the functions that are queried.

- (Linear Efficiency) The cipher-text can generated in time $\tilde{O}(k \cdot n_{\mathsf{PHFE}_1})$ group operations.

Such a scheme was recently constructed in [AGW20]. We give an overview of this scheme in Section 6.2.3.

As a result of this, the construction follows the same outline as in the case of quadratic functions replacing inner product encryption with such a partially hiding functional encryption. We directly give the formal construction below:

$\mathsf{PP} \leftarrow \mathsf{PPGen}(1^\lambda)$: Compute $(e, \mathbb{G}, \mathbb{G}_T, p, g, g_T) \leftarrow \mathcal{G}(1^\lambda)$. Set and output $\mathsf{PP} = (e, \mathbb{G}, \mathbb{G}_T, p, g, g_T)$.

$\mathsf{PP} \leftarrow \mathsf{Setup}(d, 1^{n_{\mathsf{PHFE}}}, \mathsf{PP})$: Sample random matrices $\mathbf{A}_1, \mathbf{A}_2 \leftarrow \mathbb{Z}_p^{k \times n_{\mathsf{PHFE}}}$. Generate

$$\mathbf{M} = \begin{bmatrix} \mathbf{A}_1 \otimes \mathbf{I}_{n_{\mathsf{PHFE}}} \\ \mathbf{I}_{n_{\mathsf{PHFE}}} \otimes \mathbf{A}_2 \end{bmatrix}.$$

Then, run $\mathsf{PHFE}_1.\mathsf{Setup}(d, 1^{2 \cdot k \cdot n_{\mathsf{PHFE}}}, \mathsf{PP}, [\mathbf{M}]) \rightarrow (\mathsf{PHFE}_1.\mathsf{PK}, \mathsf{PHFE}_1.\mathsf{MSK})$. Set $\mathsf{PK} = (\mathsf{PHFE}_1.\mathsf{PK}, [\mathbf{A}_1], [\mathbf{A}_2], [\mathbf{M}])$ and $\mathsf{MSK} = (\mathbf{A}_1, \mathbf{A}_2, \mathsf{PHFE}_1.\mathsf{MSK})$.

$\mathsf{CT} \leftarrow \mathsf{Enc}(\mathsf{PK}, (\mathsf{P}, \mathsf{S}))$: Sample vectors $\mathbf{v}_1, \mathbf{v}_2 \leftarrow \mathbb{Z}_p^{n_{\mathsf{PHFE}}}$. Compute $[\mathbf{z}_1] = [\mathbf{v}_1 \mathbf{A}_1 + \mathsf{S}]$ and $[\mathbf{z}_2] = [\mathbf{v}_2 \mathbf{A}_2 + \mathsf{S}]$. Compute $\mathsf{PHFE}_1.\mathsf{CT} \leftarrow \mathsf{PHFE}_1.\mathsf{Enc}(\mathsf{PHFE}_1.\mathsf{PK}, \mathsf{P}, [\mathbf{v}_1 \otimes \mathsf{S} || \mathbf{z}_1 \otimes \mathbf{v}_2])$. Output $\mathsf{CT} = (\mathsf{P}, [\mathbf{z}_1], [\mathbf{z}_2], \mathsf{PHFE}_1.\mathsf{CT})$.

$\mathsf{SK}_f \leftarrow \mathsf{KeyGen}(\mathsf{MSK}, f)$: Compute $\mathsf{PHFE}_1.\mathsf{SK}_f \leftarrow \mathsf{PHFE}_1.\mathsf{KeyGen}(\mathsf{PHFE}_1.\mathsf{MSK}, f)$. Output $\mathsf{SK}_f = \mathsf{PHFE}_1.\mathsf{SK}_f$.

$\mathsf{out} \leftarrow \mathsf{Dec}(\mathsf{SK}_f, \mathsf{CT})$: Parse $\mathsf{CT} = (\mathsf{P}, [\mathbf{z}_1], [\mathbf{z}_2], \mathsf{PHFE}_1.\mathsf{CT})$ and $\mathsf{SK}_f = \mathsf{PHFE}_1.\mathsf{SK}_f$. Compute $(\mathbf{f}(\mathsf{P}))^\top \in \mathbb{Z}_p^{{n_{\mathsf{PHFE}}}^2 \times 1}$, the vector of coefficients, where each coordinate is a degree d polynomial in P. Compute $[\alpha_1]_T = [(\mathbf{z}_1 \otimes \mathbf{z}_2)(\mathbf{f}(\mathsf{P}))^\top]_T$ and $[\alpha_2]_T = \mathsf{PHFE}_1.\mathsf{Dec}(\mathsf{PHFE}_1.\mathsf{SK}_f, \mathsf{PHFE}_1.\mathsf{CT})$. Output $[\alpha_1 - \alpha_2]_T$.

We now argue correctness:

Correctness. Observe that:

$$\begin{aligned}(\mathbf{v}_1\mathbf{A}_1 + \mathsf{S}) \otimes (\mathbf{v}_2\mathbf{A}_2 + \mathsf{S}) &= \mathsf{S} \otimes \mathsf{S} + \mathbf{v}_1\mathbf{A}_1 \otimes \mathbf{z}_2 + \mathsf{S} \otimes \mathbf{v}_2\mathbf{A}_2 \\ &= \mathsf{S} \otimes \mathsf{S} + (\mathbf{v}_1 \otimes \mathbf{z}_2)(\mathbf{A}_1 \otimes \mathbf{I}_{n_{\mathsf{PHFE}}}) + (\mathsf{S} \otimes \mathbf{v}_2)(\mathbf{I}_{n_{\mathsf{PHFE}}} \otimes \mathbf{A}_2) \\ &= \mathsf{S} \otimes \mathsf{S} + (\mathbf{v}_1 \otimes \mathbf{z}_2 || \mathsf{S} \otimes \mathbf{v}_2)\mathbf{M}\end{aligned}$$

where the second equality follows from the mixed product property of tensor products. Multiplying the equation with $(\mathbf{f}(\mathsf{P}))^\top$ and rearranging terms we get:

$$(\mathsf{S} \otimes \mathsf{S})(\mathbf{f}(\mathsf{P}))^\top = (\mathbf{v}_1\mathbf{A}_1 + \mathsf{S}) \otimes (\mathbf{v}_2\mathbf{A}_2 + \mathsf{S})(\mathbf{f}(\mathsf{P}))^\top - (\mathbf{v}_1 \otimes \mathbf{z}_2 || \mathsf{S} \otimes \mathbf{v}_2)\mathbf{M}(\mathbf{f}(\mathsf{P}))^\top$$

Note that due to correctness of the PHFE_1 scheme $\alpha_2 = (\mathbf{v}_1 \otimes \mathbf{z}_2 || \mathsf{S} \otimes \mathbf{v}_2)\mathbf{M}(\mathbf{f}(\mathsf{P}))^\top$. Because of this, the correctness holds.

Linear Efficiency. Note that the linear effciency holds due to the linear efficiency of PHFE_1. The time to compute $[\mathbf{z}_1], [\mathbf{z}_2]$ is $\tilde{O}(k \cdot n_{\mathsf{PHFE}})$ where $\tilde{O}$ hides polynomial factors in λ and $\log_2 p$. $[\mathbf{v}_1 \otimes \mathsf{S} || \mathbf{z}_1 \otimes \mathbf{v}_2]$ can also be computed in $\tilde{O}(k \cdot n_{\mathsf{PHFE}})$ operations. Finally, $\mathsf{PHFE.CT}_1$ can be computed in $\tilde{O}(n_{\mathsf{PHFE}} \cdot k)$ operations due to linear efficiency of PHFE_1. This completes the argument.

Security Proof. We prove the following theorem:

Theorem 6.2.1. *Assuming k-LIN holds, the PHFE scheme described above is secure. If the assumption is subexponentially secure, then so is the constructed PHFE scheme.*

Proof. We prove this by introducing a number of hybrids. The first hybrid corresponds to the real security game and the final security game corresponds to the simulator. We then argue that each hybrid is computationally indistinguishable to each other. Let (P, S) denote the challenge message output by the adversary and $\mathsf{PHFE}_1.\widetilde{\mathsf{Setup}}, \mathsf{PHFE}_1.\widetilde{\mathsf{Enc}}, \mathsf{PHFE}_1.\widetilde{\mathsf{KeyGen}}$ denote the simulation algorithms for PHFE_1.

Hybrid$_0$: This corresponds to the real game.

Hybrid$_1$: In this hybrid, we invoke the simulator of PHFE_1. Namely,

- Generate $(\mathsf{PHFE}_1.\mathsf{PK}, \mathsf{PHFE}_1.\mathsf{MSK}) \leftarrow \mathsf{PHFE}_1.\widetilde{\mathsf{Setup}}(d, 1^{2\cdot k\cdot n_{\mathsf{PHFE}}}, \mathsf{PP}, [\mathbf{M}])$,
- Generate $\mathsf{PHFE}_1.\mathsf{CT} \leftarrow \mathsf{PHFE}_1.\widetilde{\mathsf{Enc}}(\mathsf{PHFE}_1.\mathsf{PK}, \mathsf{P})$, and,
- For any queried functions f generate $\mathsf{PHFE}_1.\mathsf{SK}_f \leftarrow \mathsf{PHFE}_1.\widetilde{\mathsf{KeyGen}}(\mathsf{PHFE}_1.\mathsf{MSK}, f, [(\mathbf{v}_1 \otimes \mathbf{z}_2 || \mathsf{S} \otimes \mathbf{v}_2)\mathbf{M}(\mathbf{f}(\mathsf{P}))^\top])$,

These two hybrids are indistinguishable due to the security of PHFE_1 scheme. Since PHFE_1 is secure due to k-LIN, these hybrids are indistinguishable to k-LIN.

Hybrid$_2$: In this hybrid, we generate the keys differently. For any queried functions f generate $\mathsf{PHFE}_1.\mathsf{SK}_f \leftarrow \mathsf{PHFE}_1.\widetilde{\mathsf{KeyGen}}(\mathsf{PHFE}_1.\mathsf{MSK}, f, [(\mathbf{z}_1 \otimes \mathbf{z}_2)(\mathbf{f}(\mathsf{P}))^\top - (\mathsf{S} \otimes \mathsf{S})(\mathbf{f}(\mathsf{P}))^\top])$. The above two hybrids are identical as $[(\mathbf{z}_1 \otimes \mathbf{z}_2)(\mathbf{f}(\mathsf{P}))^\top - (\mathsf{S} \otimes \mathsf{S})(\mathbf{f}(\mathsf{P}))^\top] = [(\mathbf{v}_1 \otimes \mathbf{z}_2 || \mathsf{S} \otimes \mathbf{v}_2)\mathbf{M}(\mathbf{f}(\mathsf{P}))^\top]$.

Hybrid$_3$: In this hybrid, we replace $[\mathbf{z}_1]$ with random as opposed to generating it as $[\mathbf{v}_1\mathbf{A}_1 + \mathsf{S}]$. These hybrids are indistinguishable due to k-LIN. Note that to generate keys we need $\mathbf{z}_1$ exponentiated as $[\mathbf{z}_1]$ and $\mathbf{z}_2$ in the clear. The reduction receives $([\mathbf{A}_1], [\mathbf{u}])$ where $\mathbf{u}$ is either $\mathbf{v}_1\mathbf{A}_1$ or random. The reduction generates $[\mathbf{z}_1] = [\mathbf{u} + \mathsf{S}]$ and samples $\mathbf{v}_2, \mathbf{A}_2$ externally. The rest of the game is simulated as in the previous hybrid. It outputs whatever the adversary outputs. If $\mathbf{u}$ is $\mathbf{v}_1\mathbf{A}_1$, then the view of adversary is identical to the view in **Hybrid$_2$**. Otherwise, the view is identical to the view in **Hybrid$_3$**. The distinguishing advantage of the adversary distinguishing these hybrids is the advantage of the reduction in the k-LPN security game.

Hybrid$_4$: In this hybrid, we replace $[\mathbf{z}_2]$ with random as opposed to generating it as $[\mathbf{v}_2\mathbf{A}_2 + \mathsf{S}]$. Note that **Hybrid$_3$** is also indistinguishable to **Hybrid$_4$** due to k-LPN assumption. The reduction proceeds similarly to the argument made in the last hybrid. $\square$

Summing up. As a consequence, we have the following theorem:

Theorem 6.2.2. *Assuming there exists a constant $k \in \mathbb{N}$ for which k-LIN holds over prime order symmetric bilinear groups (Definition 6.1.2), then there exists a PHFE scheme satisfying Definition 2.4.1.*

6.2.3 Constructing PHFE_1

In this section, we describe on a high-level how to construct PHFE_1. We give an overview of the scheme that appeared in [AGW20]. Very informally speaking, the work upgrades a canonical IPE scheme to a PHFE_1 scheme using a special object:

A Useful Tool: Partial Garbling Scheme [IW14]. A partial garbling scheme is associated for the function class $\mathcal{F}_{\mathsf{pg}}$, that consists of functions f of the form:

$$f(\mathsf{P},\mathsf{S}) = \mathsf{S} \cdot \mathbf{f}(\mathsf{P})^{\top},$$

where $\mathbf{f}(\mathsf{P})$ is a vector whose components are constant degree d polynomials in P.

Such a scheme takes as input $f \in \mathcal{F}_{\mathsf{pg}}$, and outputs in polynomial time an affine function $\mathbf{p}_f(\mathsf{P},\mathsf{S},\mathbf{t}) = (\mathsf{S} - \underline{\mathbf{t}}||\mathbf{t}(\mathbf{L}_1(\mathsf{P} \otimes \mathbf{I}_m) + \mathbf{L}_0))$, where $\mathbf{L}_0 \in \mathbb{Z}_p^{t\times mn}$, $\mathbf{L}_1 \in \mathbb{Z}_p^{t\times m}$ depend on the function f, $\mathbf{t} \leftarrow \mathbb{Z}_p^{1\times t}$ is randomly chosen and $\underline{\mathbf{t}}$ are last n cordinates of $\mathbf{t}$. This scheme has the following properties:

- (Security) $\mathbf{p}_f(\mathsf{P},\mathsf{S},\mathbf{t})$ when $\mathbf{t} \leftarrow \mathbb{Z}_p^t$ can be simulated statistically just from $f, \mathsf{P}, f(\mathsf{P},\mathsf{S})$, and,
- (Linear Decoding) One can efficiently form a vector $\mathbf{d}_{f,\mathsf{P}}$ given f, P such that $\langle \mathbf{p}_f(\mathsf{P},\mathsf{S},\mathbf{t}), \mathbf{d}_{f,\mathsf{P}} \rangle = f(\mathsf{P},\mathsf{S})$.

Using IPE. Thus, in order to construct PHFE_1, a natural approach is to use IPE scheme to evaluate $\mathbf{p}_f(\mathsf{P},\mathsf{S},\mathbf{t})$. The construction is extremely simple except that we have a minor issue: The issue is that we need to generate "random looking" $\mathbf{t}_i$ for a given ciphertext for every function key for function f_i. However, this much randomness can't be hardcorded in either the ciphertext or the key. The idea is therefore to rely on $k - \mathsf{LIN}$ to derive fresh randomness. The ciphertext with encode a vector $[\mathbf{vA}]$ of small dimension whereas the key will encode a random matrix $[\mathbf{T}_i]$. The decryption will produce $[\mathbf{p}_{f_i}(\mathsf{P}, \mathsf{S}, \mathbf{vAT}_i)]_T$. Since the randomness is always in the exponent, it is pseudorandom due to k-LIN. The details can be found in [AGW20, Wee20].

A note on the function class. In the discussion above for PHFE, PHFE_1 as well as the partial garbling scheme we only describe the schemes handling constant degree computations in the public component. However the construction of the partial garbling scheme [IW14] actually supports a much wider class of functions in the public input (namely Arithmetic Branching Programs). As a consequence, the resulting PHFE_1 and PHFE described above (as well as in [Wee20, GJLS21]) can handle a much wider class of functions in the public input. For us even constant degree computations are enough therefore we omit this specification.

REFERENCES

[AAB15] Benny Applebaum, Jonathan Avron, and Christina Brzuska. Arithmetic cryptography: Extended abstract. In Tim Roughgarden, editor, *ITCS 2015*, pages 143–151. ACM, January 2015.

[ABDP15] Michel Abdalla, Florian Bourse, Angelo De Caro, and David Pointcheval. Simple functional encryption schemes for inner products. In Jonathan Katz, editor, *PKC 2015*, volume 9020 of *LNCS*, pages 733–751. Springer, Heidelberg, March / April 2015.

[ABJ$^+$19] Prabhanjan Ananth, Saikrishna Badrinarayanan, Aayush Jain, Nathan Manohar, and Amit Sahai. From FE combiners to secure MPC and back. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part I*, volume 11891 of *LNCS*, pages 199–228. Springer, Heidelberg, December 2019.

[ABR12] Benny Applebaum, Andrej Bogdanov, and Alon Rosen. A dichotomy for local small-bias generators. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 600–617. Springer, Heidelberg, March 2012.

[ABW10] Benny Applebaum, Boaz Barak, and Avi Wigderson. Public-key cryptography from different assumptions. In Leonard J. Schulman, editor, *42nd ACM STOC*, pages 171–180. ACM Press, June 2010.

[ACC$^+$16] Prabhanjan Ananth, Yu-Chi Chen, Kai-Min Chung, Huijia Lin, and Wei-Kai Lin. Delegating RAM computations with adaptive soundness and privacy. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 3–30. Springer, Heidelberg, October / November 2016.

[ADI$^+$17] Benny Applebaum, Ivan Damgård, Yuval Ishai, Michael Nielsen, and Lior Zichron. Secure arithmetic computation with constant computational overhead. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 223–254. Springer, Heidelberg, August 2017.

[AFH$^+$16] Martin R. Albrecht, Pooya Farshim, Dennis Hofheinz, Enrique Larraia, and Kenneth G. Paterson. Multilinear maps from obfuscation. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A, Part I*, volume 9562 of *LNCS*, pages 446–473. Springer, Heidelberg, January 2016.

[Agr19] Shweta Agrawal. Indistinguishability obfuscation without multilinear maps: New methods for bootstrapping and instantiation. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 191–225. Springer, Heidelberg, May 2019.

[AGW20] Michel Abdalla, Junqing Gong, and Hoeteck Wee. Functional encryption for attribute-weighted sums from k-Lin. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2020, Part I*, LNCS, pages 685–716. Springer, Heidelberg, August 2020.

[AIK04] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in NC^0. In *45th FOCS*, pages 166–175. IEEE Computer Society Press, October 2004.

[AIK06] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Computationally private randomizing polynomials and their applications. *Computational Complexity*, 15(2):115–162, 2006.

[AJ15] Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 308–326. Springer, Heidelberg, August 2015.

[AJL$^+$19] Prabhanjan Ananth, Aayush Jain, Huijia Lin, Christian Matt, and Amit Sahai. Indistinguishability obfuscation without multilinear maps: New paradigms via low degree weak pseudorandomness and security amplification. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 284–332. Springer, Heidelberg, August 2019.

[AJS15] Prabhanjan Ananth, Abhishek Jain, and Amit Sahai. Achieving compactness generically: Indistinguishability obfuscation from non-compact functional encryption. *IACR Cryptol. ePrint Arch.*, 2015:730, 2015.

[AJS18] Prabhanjan Ananth, Aayush Jain, and Amit Sahai. Indistinguishability obfuscation without multilinear maps: io from lwe, bilinear maps, and weak pseudorandomness. *IACR Cryptology ePrint Archive*, 2018:615, 2018.

[AK19] Benny Applebaum and Eliran Kachlon. Sampling graphs without forbidden subgraphs and unbalanced expanders with negligible error. In David Zuckerman, editor, *60th FOCS*, pages 171–179. IEEE Computer Society Press, November 2019.

[AKS83] Miklós Ajtai, János Komlós, and Endre Szemerédi. An $O(n \log n)$ sorting network. In *15th ACM STOC*, pages 1–9. ACM Press, April 1983.

[AKV04] Tim Abbot, Daniel Kane, and Paul Valiant. On algorithms for nash equilibria. unpublished manuscript, 2004. `https://web.mit.edu/tabbott/Public/final.pdf`.

[AL16] Benny Applebaum and Shachar Lovett. Algebraic attacks against random local functions and their countermeasures. In Daniel Wichs and Yishay Mansour, editors, *48th ACM STOC*, pages 1087–1100. ACM Press, June 2016.

[AL18] Prabhanjan Ananth and Alex Lombardi. Succinct garbling schemes from functional encryption through a local simulation paradigm. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part II*, volume 11240 of *LNCS*, pages 455–472. Springer, Heidelberg, November 2018.

[Ale03] Michael Alekhnovich. More on average case vs approximation complexity. In *44th FOCS*, pages 298–307. IEEE Computer Society Press, October 2003.

[ALS16] Shweta Agrawal, Benoît Libert, and Damien Stehlé. Fully secure functional encryption for inner products, from standard assumptions. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 333–362. Springer, Heidelberg, August 2016.

[AMP19] Navid Alamati, Hart Montgomery, and Sikhar Patranabis. Symmetric primitives with structured secrets. Cryptology ePrint Archive, Report 2019/608, 2019. `https://eprint.iacr.org/2019/608`.

[AP20] Shweta Agrawal and Alice Pellet-Mary. Indistinguishability obfuscation without maps: Attacks and fixes for noisy linear FE. In Vincent Rijmen and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, LNCS, pages 110–140. Springer, Heidelberg, May 2020.

[App12] Benny Applebaum. Pseudorandom generators with long stretch and low locality from random local one-way functions. In Howard J. Karloff and Toniann Pitassi, editors, *44th ACM STOC*, pages 805–816. ACM Press, May 2012.

[App13] Benny Applebaum. Pseudorandom generators with long stretch and low locality from random local one-way functions. *SIAM J. Comput.*, 42(5):2008–2037, 2013.

[AS17] Prabhanjan Ananth and Amit Sahai. Projective arithmetic functional encryption and indistinguishability obfuscation from degree-5 multilinear maps. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 152–181. Springer, Heidelberg, April / May 2017.

[BBKK18] Boaz Barak, Zvika Brakerski, Ilan Komargodski, and Pravesh K. Kothari. Limits on low-degree pseudorandom generators (or: Sum-of-squares meets program obfuscation). In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 649–679. Springer, Heidelberg, April / May 2018.

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 41–55. Springer, Heidelberg, August 2004.

[BCG$^+$19] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 291–308. ACM Press, November 2019.

[BCG$^+$20] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Correlated pseudorandom functions from variable-density LPN. In *61st FOCS*, pages 1069–1080. IEEE Computer Society Press, 2020.

[BCGI18] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 896–912. ACM Press, October 2018.

[BFKL94] Avrim Blum, Merrick L. Furst, Michael J. Kearns, and Richard J. Lipton. Cryptographic primitives based on hard learning problems. In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 278–291. Springer, Heidelberg, August 1994.

[BGdMM05] Lucas Ballard, Matthew Green, Breno de Medeiros, and Fabian Monrose. Correlation-resistant storage via keyword-searchable encryption. *IACR Cryptol. ePrint Arch.*, 2005:417, 2005.

[BGI$^+$01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 1–18. Springer, Heidelberg, August 2001.

[BGK$^+$14] Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Protecting obfuscation against algebraic attacks. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 221–238. Springer, Heidelberg, May 2014.

[BGL$^+$15] Nir Bitansky, Sanjam Garg, Huijia Lin, Rafael Pass, and Sidharth Telang. Succinct randomized encodings and their applications. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th ACM STOC*, pages 439–448. ACM Press, June 2015.

[BGMS15] Dan Boneh, Divya Gupta, Ilya Mironov, and Amit Sahai. Hosting services on an untrusted cloud. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 404–436. Springer, Heidelberg, April 2015.

[BHJ$^+$19] Boaz Barak, Samuel B. Hopkins, Aayush Jain, Pravesh Kothari, and Amit Sahai. Sum-of-squares meets program obfuscation, revisited. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 226–250. Springer, Heidelberg, May 2019.

[BIJ$^+$20] James Bartusek, Yuval Ishai, Aayush Jain, Fermi Ma, Amit Sahai, and Mark Zhandry. Affine determinant programs: A framework for obfuscation and witness encryption. In Thomas Vidick, editor, *ITCS 2020*, volume 151, pages 82:1–82:39. LIPIcs, January 2020.

[BJK15] Allison Bishop, Abhishek Jain, and Lucas Kowalczyk. Function-hiding inner product encryption. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 470–491. Springer, Heidelberg, November / December 2015.

[BK20] Anne Broadbent and Raza Ali Kazmi. Indistinguishability obfuscation for quantum circuits of low t-count. *IACR Cryptol. ePrint Arch.*, 2020:639, 2020.

[BKKV10] Zvika Brakerski, Yael Tauman Kalai, Jonathan Katz, and Vinod Vaikuntanathan. Overcoming the hole in the bucket: Public-key cryptography resilient to continual memory leakage. In *51st FOCS*, pages 501–510. IEEE Computer Society Press, October 2010.

[BLMZ19] James Bartusek, Tancrède Lepoint, Fermi Ma, and Mark Zhandry. New techniques for obfuscating conjunctions. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 636–666. Springer, Heidelberg, May 2019.

[BM21] James Bartusek and Giulio Malavolta. Candidate obfuscation of null quantum circuits and witness encryption for QMA. *IACR Cryptol. ePrint Arch.*, 2021:421, 2021.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *22nd ACM STOC*, pages 503–513. ACM Press, May 1990.

[BMSZ16] Saikrishna Badrinarayanan, Eric Miles, Amit Sahai, and Mark Zhandry. Post-zeroizing obfuscation: New mathematical tools, and the case of evasive circuits. In *Advances in Cryptology - EUROCRYPT*, pages 764–791, 2016.

[BNPW16] Nir Bitansky, Ryo Nishimaki, Alain Passelègue, and Daniel Wichs. From cryptomania to obfustopia through secret-key functional encryption. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 391–418. Springer, Heidelberg, October / November 2016.

[BPR15] Nir Bitansky, Omer Paneth, and Alon Rosen. On the cryptographic hardness of finding a Nash equilibrium. In Venkatesan Guruswami, editor, *56th FOCS*, pages 1480–1498. IEEE Computer Society Press, October 2015.

[BQ09] Andrej Bogdanov and Youming Qiao. On the security of goldreich's one-way function. In Irit Dinur, Klaus Jansen, Joseph Naor, and José D. P. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 12th International Workshop, APPROX 2009, and 13th International Workshop, RANDOM 2009, Berkeley, CA, USA, August 21-23, 2009. Proceedings*, volume 5687 of *Lecture Notes in Computer Science*, pages 392–405. Springer, 2009.

[BQ12] Andrej Bogdanov and Youming Qiao. On the security of goldreich's one-way function. *Comput. Complex.*, 21(1):83–127, 2012.

[BR14] Zvika Brakerski and Guy N. Rothblum. Virtual black-box obfuscation for all circuits via generic graded encoding. In *TCC*, pages 1–25, 2014.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In Rafail Ostrovsky, editor, *52nd FOCS*, pages 97–106. IEEE Computer Society Press, October 2011.

[BV15] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. In Venkatesan Guruswami, editor, *56th FOCS*, pages 171–190. IEEE Computer Society Press, October 2015.

[BZ14] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 480–499. Springer, Heidelberg, August 2014.

[CCC$^+$15] Yu-Chi Chen, Sherman S. M. Chow, Kai-Min Chung, Russell W. F. Lai, Wei-Kai Lin, and Hong-Sheng Zhou. Cryptography for parallel RAM from indistinguishability obfuscation. Cryptology ePrint Archive, Report 2015/406, 2015. `https://eprint.iacr.org/2015/406`.

[CCH$^+$19] Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N. Rothblum, Ron D. Rothblum, and Daniel Wichs. Fiat-Shamir: from practice to theory. In Moses Charikar and Edith Cohen, editors, *51st ACM STOC*, pages 1082–1090. ACM Press, June 2019.

[CCHR16] Ran Canetti, Yilei Chen, Justin Holmgren, and Mariana Raykova. Adaptive succinct garbled RAM or: How to delegate your database. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 61–90. Springer, Heidelberg, October / November 2016.

[CCR16] Ran Canetti, Yilei Chen, and Leonid Reyzin. On the correlation intractability of obfuscated pseudorandom functions. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A, Part I*, volume 9562 of *LNCS*, pages 389–415. Springer, Heidelberg, January 2016.

[CDM$^+$18] Geoffroy Couteau, Aurélien Dupin, Pierrick Méaux, Mélissa Rossi, and Yann Rotella. On the concrete security of Goldreich's pseudorandom generator. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 96–124. Springer, Heidelberg, December 2018.

[CEMT09] James Cook, Omid Etesami, Rachel Miller, and Luca Trevisan. Goldreich's one-way function candidate and myopic backtracking algorithms. In Omer Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 521–538. Springer, Heidelberg, March 2009.

[CH16] Ran Canetti and Justin Holmgren. Fully succinct garbled RAM. In Madhu Sudan, editor, *ITCS 2016*, pages 169–178. ACM, January 2016.

[CHJV15] Ran Canetti, Justin Holmgren, Abhishek Jain, and Vinod Vaikuntanathan. Succinct garbling and indistinguishability obfuscation for RAM programs. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th ACM STOC*, pages 429–437. ACM Press, June 2015.

[CHK$^+$19a] Arka Rai Choudhuri, Pavel Hubácek, Chethan Kamath, Krzysztof Pietrzak, Alon Rosen, and Guy N. Rothblum. Finding a nash equilibrium is no easier than breaking Fiat-Shamir. In Moses Charikar and Edith Cohen, editors, *51st ACM STOC*, pages 1103–1114. ACM Press, June 2019.

[CHK$^+$19b] Arka Rai Choudhuri, Pavel Hubacek, Chethan Kamath, Krzysztof Pietrzak, Alon Rosen, and Guy N. Rothblum. PPAD-hardness via iterated squaring modulo a composite. Cryptology ePrint Archive, Report 2019/667, 2019. `https://eprint.iacr.org/2019/667`.

[CHL$^+$15] Jung Hee Cheon, Kyoohyung Han, Changmin Lee, Hansol Ryu, and Damien Stehlé. Cryptanalysis of the multilinear map over the integers. In *EUROCRYPT*, 2015.

[CLP15] Kai-Min Chung, Huijia Lin, and Rafael Pass. Constant-round concurrent zero-knowledge from indistinguishability obfuscation. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 287–307. Springer, Heidelberg, August 2015.

[CLR15] Jung Hee Cheon, Changmin Lee, and Hansol Ryu. Cryptanalysis of the new clt multilinear maps. Cryptology ePrint Archive, Report 2015/934, 2015. `http://eprint.iacr.org/`.

[CLT13] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Practical multilinear maps over the integers. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 476–493. Springer, Heidelberg, August 2013.

[CLTV15] Ran Canetti, Huijia Lin, Stefano Tessaro, and Vinod Vaikuntanathan. Obfuscation of probabilistic circuits and applications. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 468–497. Springer, Heidelberg, March 2015.

[CM01] Mary Cryan and Peter Bro Miltersen. On pseudorandom generators in NC. In Jiri Sgall, Ales Pultr, and Petr Kolman, editors, *Mathematical Foundations of Computer Science 2001, 26th International Symposium, MFCS 2001 Marianske Lazne, Czech Republic, August 27-31, 2001, Proceedings*, volume 2136 of *Lecture Notes in Computer Science*, pages 272–284. Springer, 2001.

[CPP20] Ran Canetti, Sunoo Park, and Oxana Poburinnaya. Fully deniable interactive encryption. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2020, Part I*, LNCS, pages 807–835. Springer, Heidelberg, August 2020.

[DGG$^+$16] Nico Döttling, Sanjam Garg, Divya Gupta, Peihan Miao, and Pratyay Mukherjee. Obfuscation from low noise multilinear maps. *IACR Cryptology ePrint Archive*, 2016:599, 2016.

[DGN$^+$17] Nico Döttling, Satrajit Ghosh, Jesper Buus Nielsen, Tobias Nilges, and Roberto Trifiletti. TinyOLE: Efficient actively secure two-party computation from oblivious linear function evaluation. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 2263–2276. ACM Press, October / November 2017.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[EFKP20] Naomi Ephraim, Cody Freitag, Ilan Komargodski, and Rafael Pass. Continuous verifiable delay functions. In Vincent Rijmen and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, LNCS, pages 125–154. Springer, Heidelberg, May 2020.

[EHK$^+$13] Alex Escala, Gottfried Herold, Eike Kiltz, Carla Ràfols, and Jorge Villar. An algebraic framework for Diffie-Hellman assumptions. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 129–147. Springer, Heidelberg, August 2013.

[FHHL18] Pooya Farshim, Julia Hesse, Dennis Hofheinz, and Enrique Larraia. Graded encoding schemes from obfuscation. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part II*, volume 10770 of *LNCS*, pages 371–400. Springer, Heidelberg, March 2018.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009.

[GGH13a] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 1–17. Springer, Heidelberg, May 2013.

[GGH$^+$13b] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th FOCS*, pages 40–49. IEEE Computer Society Press, October 2013.

[GGH15] Craig Gentry, Sergey Gorbunov, and Shai Halevi. Graph-induced multilinear maps from lattices. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 498–527. Springer, Heidelberg, March 2015.

[Gil52] E. N. Gilbert. A comparison of signalling alphabets. *The Bell System Technical Journal*, 31(3):504–522, 1952.

[GJK18] Craig Gentry, Charanjit S. Jutla, and Daniel Kane. Obfuscation using tensor products. *Electronic Colloquium on Computational Complexity (ECCC)*, 25:149, 2018.

[GJLS21] Romain Gay, Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from simple-to-state hard problems: New assumptions, new techniques, and simplification. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part III*, volume 12698 of *Lecture Notes in Computer Science*, pages 97–126. Springer, 2021.

[GKP$^+$13] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 555–564. ACM Press, June 2013.

[GNN17] Satrajit Ghosh, Jesper Buus Nielsen, and Tobias Nilges. Maliciously secure oblivious linear function evaluation with constant overhead. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 629–659. Springer, Heidelberg, December 2017.

[Gol00] Oded Goldreich. Candidate one-way functions based on expander graphs. *Electronic Colloquium on Computational Complexity (ECCC)*, 7(90), 2000.

[GPS16] Sanjam Garg, Omkant Pandey, and Akshayaram Srinivasan. Revisiting the cryptographic hardness of finding a nash equilibrium. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 579–604. Springer, Heidelberg, August 2016.

[GR07] Shafi Goldwasser and Guy N. Rothblum. On best-possible obfuscation. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 194–213. Springer, Heidelberg, February 2007.

[GS08] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 415–432. Springer, Heidelberg, April 2008.

[GS16] Sanjam Garg and Akshayaram Srinivasan. Single-key to multi-key functional encryption with polynomial loss. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 419–442. Springer, Heidelberg, October / November 2016.

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In Lance Fortnow and Salil P. Vadhan, editors, *43rd ACM STOC*, pages 99–108. ACM Press, June 2011.

[HJ15] Yupu Hu and Huiwen Jia. Cryptanalysis of GGH map. *IACR Cryptology ePrint Archive*, 2015:301, 2015.

[HY17] Pavel Hubácek and Eylon Yogev. Hardness of continuous local search: Query complexity and cryptographic lower bounds. In Philip N. Klein, editor, *28th SODA*, pages 1352–1371. ACM-SIAM, January 2017.

[IKOS08] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Cryptography with constant computational overhead. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 433–442. ACM Press, May 2008.

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 572–591. Springer, Heidelberg, August 2008.

[IW14] Yuval Ishai and Hoeteck Wee. Partial garbling schemes and their applications. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *ICALP 2014, Part I*, volume 8572 of *LNCS*, pages 650–662. Springer, Heidelberg, July 2014.

[JKKZ20] Ruta Jawale, Yael Tauman Kalai, Dakshita Khurana, and Rachel Zhang. SNARGs for bounded depth computations and PPAD hardness from sub-exponential LWE. Cryptology ePrint Archive, Report 2020/980, 2020. `https://eprint.iacr.org/2020/980`.

[JLMS19] Aayush Jain, Huijia Lin, Christian Matt, and Amit Sahai. How to leverage hardness of constant-degree expanding polynomials overa $\mathbb{R}$ to build $i\mathcal{O}$. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 251–281. Springer, Heidelberg, May 2019.

[JLS21a] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 60–73. ACM, 2021.

[JLS21b] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation without lattices. *Unpublished Manuscript*, 2021.

[JMS20] Aayush Jain, Nathan Manohar, and Amit Sahai. Combiners for functional encryption, unconditionally. In Vincent Rijmen and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, LNCS, pages 141–168. Springer, Heidelberg, May 2020.

[JR13] Charanjit S. Jutla and Arnab Roy. Shorter quasi-adaptive NIZK proofs for linear subspaces. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part I*, volume 8269 of *LNCS*, pages 1–20. Springer, Heidelberg, December 2013.

[KLW15] Venkata Koppula, Allison Bishop Lewko, and Brent Waters. Indistinguishability obfuscation for turing machines with unbounded memory. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th ACM STOC*, pages 419–428. ACM Press, June 2015.

[KMN$^+$14] Ilan Komargodski, Tal Moran, Moni Naor, Rafael Pass, Alon Rosen, and Eylon Yogev. One-way functions and (im)perfect obfuscation. In *55th FOCS*, pages 374–383. IEEE Computer Society Press, October 2014.

[KMOW17] Pravesh K. Kothari, Ryuhei Mori, Ryan O'Donnell, and David Witmer. Sum of squares lower bounds for refuting any CSP. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *49th ACM STOC*, pages 132–145. ACM Press, June 2017.

[KNT18] Fuyuki Kitagawa, Ryo Nishimaki, and Keisuke Tanaka. Obfustopia built on secret-key functional encryption. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 603–648. Springer, Heidelberg, April / May 2018.

[KNY14] Ilan Komargodski, Moni Naor, and Eylon Yogev. Secret-sharing for NP. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 254–273. Springer, Heidelberg, December 2014.

[KRR17] Yael Tauman Kalai, Guy N. Rothblum, and Ron D. Rothblum. From obfuscation to the security of Fiat-Shamir for proofs. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 224–251. Springer, Heidelberg, August 2017.

[KRS15] Dakshita Khurana, Vanishree Rao, and Amit Sahai. Multi-party key exchange for unbounded parties from indistinguishability obfuscation. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 52–75. Springer, Heidelberg, November / December 2015.

[KS17] Ilan Komargodski and Gil Segev. From minicrypt to obfustopia via private-key functional encryption. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 122–151. Springer, Heidelberg, April / May 2017.

[Lin16] Huijia Lin. Indistinguishability obfuscation from constant-degree graded encoding schemes. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part I*, volume 9665 of *LNCS*, pages 28–57. Springer, Heidelberg, May 2016.

[Lin17] Huijia Lin. Indistinguishability obfuscation from SXDH on 5-linear maps and locality-5 PRGs. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 599–629. Springer, Heidelberg, August 2017.

[LM16] Baiyu Li and Daniele Micciancio. Compactness vs collusion resistance in functional encryption. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 443–468. Springer, Heidelberg, October / November 2016.

[LM18] Huijia Lin and Christian Matt. Pseudo flawed-smudging generators and their application to indistinguishability obfuscation. *IACR Cryptology ePrint Archive*, 2018:646, 2018.

[LPST16] Huijia Lin, Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation with non-trivial efficiency. In *IACR International Workshop on Public Key Cryptography*, pages 447–462. Springer, 2016.

[LT17] Huijia Lin and Stefano Tessaro. Indistinguishability obfuscation from trilinear maps and block-wise local PRGs. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 630–660. Springer, Heidelberg, August 2017.

[LV16] Huijia Lin and Vinod Vaikuntanathan. Indistinguishability obfuscation from DDH-like assumptions on constant-degree graded encodings. In Irit Dinur, editor, *57th FOCS*, pages 11–20. IEEE Computer Society Press, October 2016.

[LV17] Alex Lombardi and Vinod Vaikuntanathan. Limits on the locality of pseudorandom generators and applications to indistinguishability obfuscation. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 119–137. Springer, Heidelberg, November 2017.

[LV20] Alex Lombardi and Vinod Vaikuntanathan. Fiat-shamir for repeated squaring with applications to PPAD-hardness and VDFs. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2020, Part III*, LNCS, pages 632–651. Springer, Heidelberg, August 2020.

[MF15] Brice Minaud and Pierre-Alain Fouque. Cryptanalysis of the new multilinear map over the integers. Cryptology ePrint Archive, Report 2015/941, 2015. `http://eprint.iacr.org/`.

[MM11] Daniele Micciancio and Petros Mol. Pseudorandom knapsacks and the sample complexity of LWE search-to-decision reductions. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 465–484. Springer, Heidelberg, August 2011.

[MP13] Daniele Micciancio and Chris Peikert. Hardness of SIS and LWE with small parameters. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 21–39. Springer, Heidelberg, August 2013.

[MST03] Elchanan Mossel, Amir Shpilka, and Luca Trevisan. On e-biased generators in NC0. In *44th FOCS*, pages 136–145. IEEE Computer Society Press, October 2003.

[MSZ16] Eric Miles, Amit Sahai, and Mark Zhandry. Annihilation attacks for multilinear maps: Cryptanalysis of indistinguishability obfuscation over GGH13. In *Advances in Cryptology - CRYPTO*, 2016.

[NS92] Noam Nisan and Mario Szegedy. On the degree of Boolean functions as real polynomials. In *24th ACM STOC*, pages 462–467. ACM Press, May 1992.

[OT10] Tatsuaki Okamoto and Katsuyuki Takashima. Fully secure functional encryption with general relations from the decisional linear assumption. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 191–208. Springer, Heidelberg, August 2010.

[OW14] Ryan O'Donnell and David Witmer. Goldreich's PRG: evidence for near-optimal polynomial stretch. In *IEEE 29th Conference on Computational Complexity, CCC 2014, Vancouver, BC, Canada, June 11-13, 2014*, pages 1–12. IEEE Computer Society, 2014.

[PF79] Nicholas Pippenger and Michael J. Fischer. Relations among complexity measures. *J. ACM*, 26(2):361–381, 1979.

[PST14] Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation from semantically-secure multilinear encodings. In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, pages 500–517, 2014.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, pages 84–93, 2005.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *46th ACM STOC*, pages 475–484. ACM Press, May / June 2014.

[Tom19] Junichi Tomida. Tightly secure inner product functional encryption: Multi-input and function-hiding constructions. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part III*, volume 11923 of *LNCS*, pages 459–488. Springer, Heidelberg, December 2019.

[Vai20] Vinod Vaikunthananthan. Cs 294 lecture 4: Worst-case to average-case reductions for lwe. Class at UC Berkeley, 2020. `http://people.csail.mit.edu/vinodv/CS294/lecture4.pdf`.

[Val76] Leslie G. Valiant. Universal circuits (preliminary report). In Ashok K. Chandra, Detlef Wotschke, Emily P. Friedman, and Michael A. Harrison, editors, *Proceedings of the 8th Annual ACM Symposium on Theory of Computing, May 3-5, 1976, Hershey, Pennsylvania, USA*, pages 196–203. ACM, 1976.

[Var57] Rom Varshamov. Estimate of the number of signals in error correcting codes. *Dokl. Akad. Nauk SSSR*, 1957.

[vDGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In Henri Gilbert, editor, *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings*, volume 6110 of *Lecture Notes in Computer Science*, pages 24–43. Springer, 2010.

[Wee20] Hoeteck Wee. Functional encryption for quadratic functions from k-lin, revisited. In *TCC 2020, Part I*, LNCS, pages 210–228. Springer, Heidelberg, March 2020.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 1986.

[YYHK14] Takashi Yamakawa, Shota Yamada, Goichiro Hanaoka, and Noboru Kunihiro. Self-bilinear map on unknown order groups from indistinguishability obfuscation and its applications. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 90–107. Springer, Heidelberg, August 2014.