Indistinguishability Obfuscation from LPN over $\mathbb{F}_p$, DLIN, and PRGs in NC^0

Aayush Jain[*] Huijia Lin[†] Amit Sahai[‡]

Abstract

In this work, we study what minimal sets of assumptions suffice for constructing indistinguishability obfuscation ($i\mathcal{O}$). We prove:

Theorem(Informal): *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 aforementioned assumptions, there exists collusion resistant public-key functional encryption for all polynomial-size circuits.

This removes the reliance on the Learning With Errors (LWE) assumption from the recent work of [Jain, Lin, Sahai STOC'21]. As a consequence, we obtain the first fully homomorphic encryption scheme that does not rely on any lattice-based hardness assumption.

Our techniques feature a new notion of randomized encoding called Preprocessing Randomized Encoding (PRE) that, essentially, can be computed in the exponent of pairing groups. When combined with other new techniques, PRE gives a much more streamlined construction of $i\mathcal{O}$ while still maintaining reliance only on well-studied assumptions.

[*]UCLA, Center for Encrypted Functionalities, and NTT Research. Email: aayushjain@cs.ucla.edu.
[†]UW. Email: rachel@cs.washington.edu.
[‡]UCLA, Center for Encrypted Functionalities. Email: sahai@cs.ucla.edu.

Contents

1 Introduction

Indistinguishability obfuscation ($i\mathcal{O}$) for general programs computable in polynomial time [BGI$^+$01] enables us to hide all implementation-specific details about any program while preserving its functionality. $i\mathcal{O}$ is a fundamental and powerful primitive, with a plenthra of applications in cryptography and beyond. It is hence extremely important to investigate how to build $i\mathcal{O}$, based on as *minimal assumptions* as possible, and via as *simple constructions* as possible. Advances on understanding what assumptions imply $i\mathcal{O}$ and simplification of $i\mathcal{O}$ constructions have immediate implications on the rest of cryptography through the many applications of $i\mathcal{O}$. So far, through the accumulation of extensive research by a large community since the first mathematical candidate $i\mathcal{O}$ proposal by [GGH$^+$13] (see the survey in [GJLS21] and references therein), we recently saw the first construction of $i\mathcal{O}$ [JLS21] based on four well-studied assumptions: Learning With Errors (LWE) [Reg05], Decisional Linear assumption (DLIN) [BGdMM05] over bilinear groups, Learning Parity with Noise over $\mathbb{F}_p$ [IPS09], and Pseudo-Random Generators in NC^0 [Gol00].

While the work of Jain, Lin and Sahai [JLS21] settles the feasibility of $i\mathcal{O}$ on solid assumptions, much still awaits be answered, even on the front of feasibility. A fundamental question to study next is:

"What minimal sets of well-studied assumptions suffice to construct $i\mathcal{O}$?"

From a complexity theoretic perspective, studying the minimal sets of sufficient assumptions helps deepen our understanding of the nature and structure of $i\mathcal{O}$, as well as understanding the power of these sufficient assumptions (via the many applications of $i\mathcal{O}$). It also serves as a test-bed for new ideas and techniques, and may lead to new ways of constructing $i\mathcal{O}$ and other primitives.

As we embark upon this question, it is important to keep an open mind. The answers may not be unique – there may be different minimal combinations of assumptions that are sufficient for $i\mathcal{O}$, and we do not know what the future may bring. Perhaps LWE alone is enough, or perhaps not. The answers may not be what we expect. Unexpected answers may teach us just as much as (if not more than) the answers that confirm our expectation. Our work here presents one such answer that challenges expectations, and at the same time, simplifies the overall architecture needed to construct $i\mathcal{O}$ from well-studied assumptions.

Our Result. We improve upon the $i\mathcal{O}$ construction of [JLS21] by removing their reliance on LWE. We thus obtain $i\mathcal{O}$ based on the following three assumptions, which generates interesting consequences that we discuss below.

Theorem 1.1 (Informal)**.** 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. Assuming only polynomial security of the assumptions above yields polynomially secure functional encryption for all polynomial-size circuits.

It is interesting to note that of the three assumptions above, 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 relatively sparse errors, that is, $\delta \geq \frac{1}{2}$ in our language above [Ale03, AB15].) Thus, this work removes one, namely LWE, of the two public-key assumptions in [JLS21], making a first step towards understanding the minimal set of assumptions underlying $i\mathcal{O}$.

Lattice v.s. (pairing + LPN over $\mathbb{F}_p$ + PRG in NC^0). 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.

Corollary 1.1 (Informal)**.** Assume the same assumptions as in the Theorem 1.1. Then, fully homomorphic encryption and attribute-based encryption for all polynomial-sized circuits exist.

Beyond FHE and ABE, lattice problems and techniques have been at the heart of nearly every work over the past decade attempting to achieve advanced cryptographic feasibility goals. Our theorem shows that, through $i\mathcal{O}$, the combination of pairing, LPN over $\mathbb{F}_p$, and constant-locality PRG is just as powerful as (and potentially more powerful than) lattice techniques for achieving feasibility goals.

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.)?

Streamlining $i\mathcal{O}$ Construction. In our minds, an equally important contribution of our work is streamlining of the construction of $i\mathcal{O}$ from well-studied assumptions. Current surviving $i\mathcal{O}$ proposals are all highly complex. They usually start with building a minimal tool and then transform it to $i\mathcal{O}$ through a number of sophisticated transformations in the literature. Take the recent construction of $i\mathcal{O}$ in [JLS21] as an example. It starts with 1) building a 1-key secret-key FE scheme for NC^0 with sublinearly compact ciphertext, that is only weakly $(1-1/\mathsf{poly}(\lambda))$-secure, then 2) lift the function class to handle circuits via transformations in [AJS15b, LV16, Lin16], 3) amplify security via [GJLS21], 4) turn secret-ley FE to public-key FE via [BNPW16], 5) transform FE with sublinear-size ciphertext to FE with sublinear-time encryption [LPST16, GKP$^+$13], and finally 6) construct $i\mathcal{O}$ from public-key FE for circuits with sublinear-time encryption [AJ15, BV15]. While there exist alternative transformations for each of the steps, and other constructions of $i\mathcal{O}$ may omit some of the steps, it is a widely recognized problem that existing $i\mathcal{O}$ constructions are complex.

Our new construction of $i\mathcal{O}$, while removing the reliance of LWE, also removes the reliance on most transformations used in previous constructions. Starting from a known and simple partially hiding functional encryption scheme based on DLIN by [Wee20], we directly construct a public-key FE for circuits[1] with sublinear-time encryption, which implies $i\mathcal{O}$ via [AJ15, BV15].

To enable our results, we propose and achieve the new notion of *Preprocessed Randomized Encodings* (PRE). Roughly speaking, PRE allows for preprocessing the input $\boldsymbol{x}$ and random coins $\boldsymbol{r}$ into *preprocessed input* $(\mathsf{PI}, \mathsf{SI})$ where PI is public and SI is private, so that, later a randomized encoding of $(f, \boldsymbol{x})$ can be computed by polynomials with only degree 2 in SI, and constant degree in PI, over general prime field $\mathbb{F}_p$. PRE guarantees that the preprocessed input $(\mathsf{PI}, \mathsf{SI})$ can be computed in time sublinear in the size of the circuit f and the randomized encoding together with PI hides the actual input $\boldsymbol{x}$.

We now proceed to an overview of our techniques.

2 Technical Overview

FE Bootstrapping. A common thread in many recent $i\mathcal{O}$ constructions [AJS15b, Lin16, AS17, Lin17, LT17, AJS18, LM18, AJL$^+$19, JLMS19, JLS19, GJLS21, JLS21] is FE bootstrapping – transformations that lift FE for computing simple functions in NC^0 to full fledged functional encryption for polynomial-sized circuits. Such an FE scheme in turn implies $i\mathcal{O}$ by the works of [AJ15, BV15].

More specifically, functional encryption is an advanced form of public key encryption, which allows generating functional secret keys associated with a specific function $f : \{0,1\}^n \to \{0,1\}^m$, denoted as SK_f, such that, decrypting a ciphertext $\mathsf{CT}(\boldsymbol{x})$ using this secret key reveals only the output $f(\boldsymbol{x})$, and nothing else about the encrypted input $\boldsymbol{x}$. To imply $i\mathcal{O}$, it suffices to have FE with the following properties:

- It supports publishing a single functional decryption key SK_C, for a circuit $C : \{0,1\}^n \to \{0,1\}^m$ where every output bit is computable by a circuit of fixed size $\mathsf{poly}(\lambda)$ in the security parameter[2]. The overall size of C is $\mathsf{poly}(\lambda) \cdot m$.

- It is crucial that FE has encryption that runs in time sublinear in the size of the circuit C: $T_{\mathsf{Enc}} = \mathsf{poly}(\lambda) \cdot m^{1-\epsilon}$ – we refer to this property as **sublinear time-succinctness**.

In contrast, FE with encryption that takes time polynomial in the circuit size is just equivalent to vanilla public key encryption [SS10, GVW12]. An intermediate level of efficiency known as **sublinear size-succinctness** only requires the ciphertext size to be sublinear $|\mathsf{CT}(\boldsymbol{x})| = \mathsf{poly}(\lambda) \cdot m^{1-\epsilon}$, without restricting the encryption time. It has been shown that FE with sublinear size-succinctness in fact implies FE with sublinear time-succinctness, but additionally assuming LWE [LPST16, GKP$^+$13]. In this work, one of our technical contributions is presenting a direct way of constructing FE with sublinear *time*-succinctness without LWE.

To reach the above powerful FE via bootstrapping, we start with FE schemes supporting the most expressive class of functions that we know how to build from standard assumptions. Partially-hiding functional encryption generalizes the syntax of functional encryption to allow a public input PI that does not need to be hidden in addition to the secret input SI. Furthermore, decryption reveals only $h(\mathsf{PI}, \mathsf{SI})$, where h is the function for which the functional decryption key

[1]albeit with the constraint that every output bit is computed by a λ-size circuit. This is without loss of generality via Yao's Garbled Circuits.

[2]Our construction of FE do not place the restriction on C to have logarithmic-depth, though FE with such a restriction is also sufficient for $i\mathcal{O}$.

is generated. So far, from standard assumptions over bilinear maps of order p (e.g. DLIN), prior works [JLS19, GJLS21, Wee20] constructed PHFE for polynomials h over $\mathbb{Z}_p$ that have constant degree in the public input PI and only degree-2 in the private input SI. We say such a polynomial h has degree-$(O(1), 2)$.

$$h(\mathsf{PI}, \mathsf{SI}) = \sum_{j,k} g_{j,k}(\mathsf{PI}) \cdot x_j \cdot x_k \mod p, \quad \text{where } g_{j,k} \text{ has constant degree}$$

Furthermore, known PHFE schemes enjoy strong simulation security and their encryption runs in time linear in the length of the input: $T_{\mathsf{Enc}} = (|\mathsf{PI}| + |\mathsf{SI}|)\,\mathsf{poly}(\lambda)$. Both these properties will be instrumental later.

Perhaps the most straightforward way of bootstrapping FE for simple functions to FE for complex functions is using the former to compute a Randomized Encoding (RE) π of the complex function $C(\boldsymbol{x})$, from which the output can be derived. It seems, then, that all we need is an RE that can be securely evaluated using degree-$(O(1), 2)$ PHFE. Unfortunately, this idea immediately hits a key barrier: Known RE encoding algorithms $\mathsf{Encode}_C(\boldsymbol{x}; \boldsymbol{r})$ have at least locality 4 and hence degree 4 (over $\mathbb{Z}_p$) in $\boldsymbol{x}$ and $\boldsymbol{r}$, both of which must be kept private. Making the degree smaller has been a long-standing open question. To circumvent this issue, we formalize a new notion of degree-$(O(1), 2)$ randomized encoding that crucially relies on *input preprocessing*.

Preprocessed Randomized Encoding. The key properties of a preprocessed Randomized Encoding (PRE) scheme are: *(i)* encodings can be generated using degree-$(O(1), 2)$ polynomials h on pre-processed inputs $(\mathsf{PI}, \mathsf{SI})$; and *(ii)* the input preprocessing has *sublinear time succinctness*. More precisely, the syntax of PRE is described below.

Preprocessed Randomized Encoding

- $\mathsf{PRE.PreProc}(p, \boldsymbol{x}) \to (\mathsf{PI}, \mathsf{SI})$. The preprocessing algorithm converts an input $\boldsymbol{x} \in \{0,1\}^n$ and random tape $\boldsymbol{r}$ into a preprocessed input $(\mathsf{PI}, \mathsf{SI})$ over $\mathbb{Z}_p$. It is important that preprocessing does not depend on the circuit to be encoded later (but only an upper bound on its size). It must satisfy **sublinear time-succinctness** in the sense that preprocessing time is sublinear in the size of the computation, $T_{\mathsf{PreProc}} = m^{1-\epsilon}\,\mathsf{poly}(\lambda)$.
- $\mathsf{PRE.Encode}(C, \mathsf{PI}, \mathsf{SI}) = \boldsymbol{\pi}$. The encoding algorithm takes a circuit C of size $m\,\mathsf{poly}(\lambda)$ and a preprocessed input $(\mathsf{PI}, \mathsf{SI})$, and produces a binary randomized encoding $\boldsymbol{\pi}$. $\mathsf{PRE.Encode}_C$ can be computed by a polynomial mapping over $\mathbb{Z}_p$ with constant degree in PI and degree 2 in SI.
- $\mathsf{PRE.Decode}(\boldsymbol{\pi}) = \boldsymbol{y}$. The decoding algorithm decodes the output $\boldsymbol{y} = C(\boldsymbol{x})$ from $\boldsymbol{\pi}$.

Indistinguishability security: PRE guarantees that $(\mathsf{PI}, \boldsymbol{\pi})$ generated from $(C, \boldsymbol{x}_0)$ or $(C, \boldsymbol{x}_1)$ are indistinguishable as long as $C(\boldsymbol{x}_0) = C(\boldsymbol{x}_1)$.

If we had such PRE, we can easily construct the desired powerful FE as follows:

$$\begin{aligned} \mathsf{FE.SK}: &\quad \mathsf{PHFE.SK}_h\ , \text{ where } h(\mathsf{PI}, \mathsf{SI}) = \mathsf{PRE.Encode}_C(\mathsf{PI}, \mathsf{SI}) = \boldsymbol{\pi} \\ \mathsf{FE.CT}: &\quad \mathsf{PHFE.CT}(\mathsf{PI}, \mathsf{SI})\ , \text{ where } (\mathsf{PI}, \mathsf{SI}) \leftarrow \mathsf{PRE.PreProc}(p, \boldsymbol{x}) \end{aligned}$$

The simulation security of the underlying PHFE guarantees that the only information revealed is $\mathsf{PI}, \boldsymbol{\pi}$. Hence by the indistinguishability security of pRE, FE ciphertexts of inputs $\boldsymbol{x}_0$, $\boldsymbol{x}_1$ that produce the same outputs are indistinguishable. For *time succinctness*, since the preprocessing takes sublinear time $m^{1-\epsilon}\,\mathsf{poly}(\lambda)$ and PHFE encryption takes time proportional to the length of the preprocessed inputs, which is also sublinear, we have that FE encryption has sublinear time succinctness.

2.1 Challenges to Constructing Preprocessed Randomized Encoding

The main challenges in constructing PRE is making sure that: *(i)* encoding is only degree 2 in the private preprocessed input SI; and *(ii)* preprocessing has sublinear time-succinctness. Towards this, our starting point is to consider a known randomized encoding scheme that has constant *locality* and *sublinear randomness* (explained next), and somehow modify it so that it can enjoy degree-$(O(1), 2)$ encoding. Such a constant-degree RE scheme can be obtained by combining a constant-locality RE, such as [Yao86], with a PRG in NC^0. The encoding algorithm works as $\pi = \mathsf{Encode}'_C(\boldsymbol{x}; \boldsymbol{r}' = \mathsf{PRG}(\boldsymbol{r}))$, where the random tape has sublinear length $|\boldsymbol{r}| = m^{1-\tau} \mathsf{poly}(\lambda)$ if the Encode' algorithm uses a linear number of random coins $|\boldsymbol{r}'| = O(m) \mathsf{poly}(\lambda)$ and PRG has appropriate polynomial stretch. We call this property *sublinear randomness*; it is needed because the PRE encoding algorithm is *deterministic* and hence the sublinearly short preprocessed input $(\mathsf{PI}, \mathsf{SI})$ must encode all the randomness needed for producing the encoding. Observe that Encode' has constant locality (and hence degree) in $(\boldsymbol{x}, \boldsymbol{r})$, but the locality is much higher than 2.

High-level approach for PRE: So how can we use preprocessing to reduce the encoding degree to just 2 in private proprocessed inputs? We start by adapting several ideas from [JLS21] that were used to construct objects called structured-seed PRGs, to constructing our desired PRE. Here is the high-level approach:

- Since the public input PI is supposed to hide $\boldsymbol{x}' = \boldsymbol{x}, \boldsymbol{r}$, we will set PI as an encryption $\mathsf{HEEnc}(\boldsymbol{x}')$ of $\boldsymbol{x}'$ using a special-purpose homomorphic encryption scheme.
- We set SI to contain the secret key of this homomorphic encryption and some other "pre-processed information" about the encryption. Crucially, we need to ensure that PI, SI can be computed by a circuit of size sublinear in size of C.
- Given PI, SI, the encode algorithm Encode first takes PI and homomorphically evaluate Encode' to obtain an output encryption $\mathsf{HEEnc}(\pi)$. Then, it takes SI and decrypts $\mathsf{HEEnc}(\pi)$ to obtain π. We will ensure that homomorphic evaluation of a locality-d function Encode' is a degree d operation on PI, and crucially decryption is a degree 2 operation in SI (and has at most constant degree in $\mathsf{HEEnc}(\pi)$). Because of this, Encode will have degree-$(O(1), 2)$.

Instantiation via LPN over $\mathbb{F}_p$. An example of such a homomorphic encryption scheme is based on LPN over $\mathbb{F}_p$.

$$\mathsf{PI} = \mathsf{HEEnc}(\boldsymbol{x}') = (\mathbf{A}, \boldsymbol{b} = \boldsymbol{s}\mathbf{A} + \boldsymbol{e} + \boldsymbol{x}' \mod p)$$

where the dimension $\dim$ is polynomially related with λ, but relatively small as we describe below. We sample $\mathbf{A} \leftarrow \mathbb{Z}_p^{\dim \times |\boldsymbol{x}'|}$, $\boldsymbol{s} \leftarrow \mathbb{Z}_p^{1 \times \dim}$, and the errors $\boldsymbol{e}$ is chosen so that each coordinate is non-zero with probability $\dim^{-\delta}$ for any constant $\delta > 0$ associated with the LPN over $\mathbb{F}_p$ assumption.

To come up with SI for decryption. We observe that for every locality-d polynomial h the following equation holds:

$$h(\boldsymbol{b} - \mathbf{A}\boldsymbol{s}) = h(\boldsymbol{x}' + \boldsymbol{e})$$

The LHS of the equation tells us that if we include in SI all degree-d monomials of $\boldsymbol{s}$, namely, $\mathsf{SI} = (1||\boldsymbol{s})^{\otimes d}$, then the above quantity can be computed by a polynomial that is degree d in $\mathsf{PI} = (\mathbf{A}, \boldsymbol{b})$ and in fact linear in SI. By choosing $\dim$ to be polynomially smaller than $|\boldsymbol{x}|$, the above SI will still be sufficiently succinct for our purposes. The RHS of the equation tells us that the error $\boldsymbol{e}$ is sparse, and h depends only on a constant number d variables, and thus with probability $1 - O(\dim^{-\delta})$, we have $h(\boldsymbol{x}' + \boldsymbol{e}) = h(\boldsymbol{x}')$. This almost matches what we want, except that decryption has a noticable error probability $O(\dim^{-\delta})$.

To correct the decryption errors, the key observation is that for a polynomial mapping Encode'_C with long outputs, the error vector $\mathsf{Corr} = \mathsf{Encode}'_C(\boldsymbol{x}'+\boldsymbol{e}) - \mathsf{Encode}'_C(\boldsymbol{x}')$ is sparse: only a $O(\dim^{-\delta})$ fraction of elements in Corr are non-zero. Prior work [JLS21] developed a technique for "compressing" such sparse vector Corr into (U, V) of sublinear length $|U, V| = m^{1-\epsilon}\,\mathsf{poly}(\lambda)$. From U, V, Corr can be expanded out using only degree 2. Therefore, by adding (U, V) to SI, we can decrypt and then correct errors in the output with just degree-2 computation in U, V.

$$\mathsf{SI} = (1||\boldsymbol{s})^{\otimes d}, U, V$$

However, the compression mechanism of [JLS21] only guarantees that U, V are *size-succinct*, but are not time-succinct, and in fact, constructing them takes time linear in the circuit size m.

Barriers to time-succinctness: Unfortunately, the above approach cannot achieve time-succinctness for the following reasons: The preprocessing algorithm needs to compute the errors $\mathsf{Corr} = \mathsf{Encode}'_C(\boldsymbol{x}'+\boldsymbol{e}) - \mathsf{Encode}'_C(\boldsymbol{x}')$ in the decryption output. Though the error vector is sparse, every element could be wrong with $\Omega(\dim^{-\delta})$ probability, depending on the LPN noises $\boldsymbol{e}$ used to encrypt $\boldsymbol{x}'$ and the input-output dependency graph of the function Encode'_C computed. Therefore, the circuit implementing the preprocessing must have Encode'_C stored. This creates two problems: *(i)* the proprocessing time (in the circuit model) is at least $|C|$, and more subtly, *(ii)* the proprocessing depends on C.

In the previous work of [JLS21], they deal with the first issue by invoking the transformation from size-succinct FE to time-succinct FE [LPST16, GKP+13] assuming LWE. The second issue is not a problem, since they construct structured-seed PRG and only apply the aforementioned technique to a fixed PRG in NC^0. However, structured-seed PRG alone is not enough for FE bootstrapping, and they need to additionally rely on FHE based on LWE, and security amplification techniques which again rely on LWE.

In this work, to streamline the construction of $i\mathcal{O}$, and to weaken the underlying assumptions, we want to construct PRE that directly achieves time-succinctness. Next, we discuss how to address the first issue above using the idea of *amortization*.

Key idea: Amortization. To get around the hurdle that preprocessing a single input $\boldsymbol{x}$ seems to inherently take time proprotional to $|C|$, we ask a simpler question: can we "batch-preprocess" in sublinear time? To make it precise, say we have k input vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k$ each of dimension n, and we are interested in learning $h(\boldsymbol{x}_1), \ldots, h(\boldsymbol{x}_k)$ w.r.t. a polynomial mapping $h : \{0,1\}^n \to \{0,1\}^{m'}$ with constant locality. Can we batch-process $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k\}$ into a public and a secret input $(\mathsf{PI}, \mathsf{SI})$ in time sublinear in $m' \cdot k$, such that each $h(\boldsymbol{x}_i)$ can be computed with constant degree in PI and degree 2 in SI. Our answer is *Yes*!

Furthermore, in order to get around the subtler problem that preprocessing depends on Encode'_C, we will consider a version of amortized preprocessing for computing polynomials h that have a fixed set of monomials $\mathcal{Q} = \{Q_1, \ldots, Q_{m'}\}$. We say that $h(\boldsymbol{x}_1, \cdots, \boldsymbol{x}_k)$ has monomial pattern $\mathcal{Q}$ if it has form:

$$h(\boldsymbol{x}_1, \cdots, \boldsymbol{x}_k) = \sum_{i,j} \eta_{i,j} Q_j(\boldsymbol{x}_i) \bmod p\ , \text{ where } \eta_{i,j} \text{ are integer coefficients} \tag{1}$$

The preprocessing is then allowed to depend on the monomials $\mathcal{Q}$, but not the polynomials h to be computed later. We formalize this tool called Preprocessed Polynomial Encoding (PPE) below.

Preprocessed Polynomial Encoding

- $\mathsf{PPE.PreProc}(p, \mathcal{Q}, \boldsymbol{x}_1, \cdots, \boldsymbol{x}_k) \rightarrow (\mathsf{PI}, \mathsf{SI})$. Given a collection of constant degree-d monomials, $\mathcal{Q} = \{Q_1, \ldots, Q_{m'}\}$, the preprocessing algorithm converts a batch of k inputs $\{\boldsymbol{x}_i \in \{0,1\}^n\}_{i \in [k]}$ into a preprocessed input $(\mathsf{PI}, \mathsf{SI})$ over $\mathbb{Z}_p$. It satisfies **sublinear time-succinctness** in the sense that preprocessing time is sublinear in $m' \cdot k$.
- $\mathsf{PPE.Decode}(p, \mathcal{Q}, h, \mathsf{PI}, \mathsf{SI}) = y$. The decoding algorithm decodes the output

$$y = h(\boldsymbol{x}_1, \cdots, \boldsymbol{x}_k) = \sum_{i,j} \eta_{i,j} Q_j(\boldsymbol{x}_i) \bmod p \ .$$

Indistinguishability security: PPE guarantees that PI for any two different inputs $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_k$ and $\boldsymbol{x}'_1, \cdots, \boldsymbol{x}'_k$ are indistinguishable.

Next we need to answer two questions: *(i)* Can we construct PPE?; and *(2)* Is this amortization useful to construct PRE? Below, we answer the second question first.

Constructing PRE **using (amortized)** PPE**.** In order to construct PRE scheme, we need a randomized encoding scheme (with sublinear randomness) with an encoding algorithm Encode'_C that is exactly the kind of polynomials that PPE can handle (Equation 1). Then, we can simply use the PPE preprocessing as the PRE preprocessing. More precisely, there should exist a universal set of monomials $\mathcal{Q}$, such that, for every complex circuit C,

$$\mathsf{Encode}'_C(\boldsymbol{x}, \boldsymbol{r}) = h(\boldsymbol{x}_1, \cdots, \boldsymbol{x}_k) = \sum_{i,j} \eta_{i,j} Q_j(\boldsymbol{x}_i) \bmod p \ , \text{ where } \eta_{i,j}\text{'s depends on } C\text{, but not } Q_j\text{'s.}$$

We construct such an RE, denoted as ARE, using Yao's garbling scheme [Yao86] and a PRG in NC^0. Recall that we consider circuits $C : \{0,1\}^n \rightarrow \{0,1\}^{m=m'k}$ where every output bit is computable by a circuit of fixed size λ. We can divide such a circuit C into k chunks $C_1, \ldots, C_k$ where circuit C_i computes the i^{th} chunk of outputs of C and has size $m'\lambda$, and then we can garble each of the chunks separately.

$$\mathsf{ARE.Encode}(C, \boldsymbol{x}, \boldsymbol{r}_1, \ldots, \boldsymbol{r}_k) = \mathsf{Yao.Gb}(C_1, \boldsymbol{x}; \mathsf{PRG}(\boldsymbol{r}_1)), \ldots, \mathsf{Yao.Gb}(C_{k'}, \boldsymbol{x}; \mathsf{PRG}(\boldsymbol{r}_k)),$$

The idea is viewing $\{\boldsymbol{x}_i = (\boldsymbol{x}, \boldsymbol{r}_i)\}$ as the k inputs to be batch processed. But, do the functions $\{\mathsf{Yao.Gb}(C_i, \star, \star)\}$ share a universal set of monomials? Unfortunately, this is not the case since the computation of each garbled table depends on the gates in C_i. To solve this problem, we modify our approach to garble the universal circuit and treat C_i's as part of input to be garbled. More precisely,

$$\mathsf{ARE.Encode}(C, \boldsymbol{x}, \boldsymbol{r}_1, \ldots, \boldsymbol{r}_k) = \mathsf{Yao.Gb}(U, (C_1, \boldsymbol{x}), \mathsf{PRG}(\boldsymbol{r}_1)), \ldots, \mathsf{Yao.Gb}(U, (C_k, \boldsymbol{x}), \mathsf{PRG}(\boldsymbol{r}_k)),$$

where U is a universal circuit that takes as input $C_i, \boldsymbol{x}$ and outputs $C_i(\boldsymbol{x})$. Now the computation of the garbled tables no longer depend on C_i, neither does the input garbling of $\boldsymbol{x}$. The only part that depends on C_i is the input garbling of C_i, which looks like $(1 - C_{ij})l_0 + C_{ij}l_1$, for every bit of description of C_i. Examining more closely, we see that the monomials for computing the labels are in fact universal, and C_i only affects the coefficients that combine these monomials. This is exactly the type of polynomials that PPE can handle. More details are provided in Section 5, where we also show that the size of the garbling is linear in $|C| = m\lambda$ and the total input length $|\boldsymbol{x}' = (\boldsymbol{x}, \boldsymbol{r})|$ is sublinear in m.

2.2 Constructing Proprocessed Polynomial Encoding

We now construct our key technical tool PPE. For simplicity, in this overview, we will focus on computing just the a collection of degree d monomials $\mathcal{Q} = \{Q_i(\boldsymbol{x}_j)\}_{i\in[m'],j\in[k]}$, as it illustrates the idea behind our preprocessing procedure, and polynomials with monomial pattern $\mathcal{Q}$ can be computed in the same degree as the monomials. Similar to before, the public preprocessed input PI contains a LPN encryption of each $\boldsymbol{x}_j$, that is,

$$\mathsf{PI} = \{\mathsf{HEEnc}(\boldsymbol{x}_j) = (\mathbf{A}_j, \boldsymbol{b}_j = \boldsymbol{s}\mathbf{A}_j + \boldsymbol{e}_j + \boldsymbol{x}_j)\}_{j\in[k]},$$

where $\mathbf{A}_j \leftarrow \mathbb{Z}_p^{n\times k}$, $\boldsymbol{s} \leftarrow \mathbb{Z}_p^{1\times k}$, and $\boldsymbol{e}_j \in \mathbb{Z}_p^k$ where each coordinate is zero with probability $k^{-\delta}$. Here we set the LPN dimension to k, which is set to be polynomially related to but polynomially smaller than n. Given PI, we can homomorphically evaluate all monomials in $\mathcal{Q}$ to obtain encryption of the outputs $\{\mathsf{HEEnc}(Q_i(\boldsymbol{x}_j))\}_{i,j}$.

Next, we construct SI so that these ciphertexts can be decypted and errors can be corrected. For decryption, SI includes all degree d monomials in the secret key $\boldsymbol{s}$, $\mathsf{SI}_0 = (1||\boldsymbol{s})^{\otimes d}$, so that one can obtain the erroneous outputs $\{Q_i(\boldsymbol{x}_j + \boldsymbol{e}_j)\}_{i,j}$. Next, think of the errors Corr as arranged in a $m' \times k$ matrix, where $\mathsf{Corr}[i,j] = Q_i(\boldsymbol{x}_j + \boldsymbol{e}_j) - Q_i(\boldsymbol{x}_j)$. We do not compress the entire matrix Corr in one shot, nor compressing it column by column, the new idea is compressing it row by row. Each row, denoted by Corr_i, has dimension k and contains the errors related to computing a single monomial Q_i on all inputs $\{\boldsymbol{x}_j\}_j$,

$$\mathsf{Corr}_i = \{Q_i(\boldsymbol{x}_j + \boldsymbol{e}_j) - Q_i(\boldsymbol{x}_j)\}_{j\in[k]} \ .$$

If we can compress each Corr_i into SI_i in (amortized) sublinear time $(k^{1-\Omega(1)})\,\mathsf{poly}(\lambda)$, then the overall time for computing $\mathsf{SI} = (\mathsf{SI}_0, \mathsf{SI}_1, \cdots, \mathsf{SI}'_m)$ is $(k^{1-\Omega(1)} \cdot \frac{m}{k})\,\mathsf{poly}(\lambda)$, sublinear in $m' \cdot k$. Given such SI, we can indeed correct all errors in degree 2 and obtain the desired outputs $\{Q_i(\boldsymbol{x}_j)\}_{i,j}$.

The compressed version SI_i. So what is special about compressing each row Corr_i? The key is that elements in one row $\{\mathsf{Corr}[i,j]\}_j$ are all *independent*, because the value $\mathsf{Corr}[i,j]$ depends on $\boldsymbol{e}_j, \boldsymbol{x}_j$, which is independent for different j's. In comparison, note that this is not the case for elements in one column $\{\mathsf{Corr}[i,j]_i\}$. This is because two different monomials Q_i and $Q_{i'}$ may depend on the same input variable, say the k'th, and hence $\mathsf{Corr}[i,j]$ and $\mathsf{Corr}[i',j]$ both depend on the same noise $e_{j,k}$ used for hiding $x_{j,k}$. The independence and the fact that each element $\mathsf{Corr}[i,j]$ is non-zero with probability $O(k^{-\delta})$ imply that each row Corr_i has $O(k^{1-\delta})$ non-zero elements with overwhelming probability.

We rely on both the sparsity of and independence of elements in Corr_i to compress it. Let's first see how the compressed version SI_i looks like. We assign elements in Corr_i into $T = k^{1-\delta}$ square matrices $\{\mathbf{M}_{i,\gamma}\}_{\gamma\in[T]}$, each of size $(t = k^{\delta/2}) \times (t = k^{\delta/2})$. The assignment can be arbitrary as long as every element $\mathsf{Corr}[i,j]$ is assigned to a unique location in one of the matrices $\mathbf{M}_{i,j_1}[j_2, j_3]$. We denote by $\boldsymbol{\phi}$ this assignement, $\boldsymbol{\phi}(j) = (j_1, j_2, j_3)$. Observe that on average, each matrix $\mathbf{M}_{i,\gamma}$ contains less than 1 non-zero entries. By the independence of elements in Corr_i again, every matrix $\mathbf{M}_{i,\gamma}$ has at most λ non-zero entries, with overwhelming probability in λ. Thus, every matrix $\mathbf{M}_{i,\gamma}$ has rank less than λ and can be decomposed into $\mathbf{U}_{i,\gamma}, \mathbf{V}_{i,\gamma} \in \mathbb{Z}_p^{t\times\lambda}$ such that $\boldsymbol{M}_{i,\gamma} = \mathbf{U}_{i,\gamma} \cdot \mathbf{V}_{i,\gamma}^\top$. The compressed version $\mathsf{SI}_i = \{\mathbf{U}_{i,\gamma}, \mathbf{V}_{i,\gamma}\}_{\gamma\in[t_1]}$ contains exactly these $\mathbf{U}, \mathbf{V}$ matrices, and the value of $Q_i(\boldsymbol{x}_j)$ can be computed in degree $(O(1), 2)$ from PI, SI_0 and SI_i as follows:

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

The size of $\mathsf{SI}_i = O(T \times t \times \lambda) = O(k^{1-\delta} \times k^{\delta/2} \times \lambda) = O(k^{1-\delta/2}\lambda)$ is sublinear in k as desired.

Computing of SI_i in sublinear time. We now show that beyond being size-succinct, each SI_i can also be computed in time sublinear in k in an amortized fashion. More precisely, we show that the collection of $\mathsf{SI}_1, \ldots, \mathsf{SI}_{m'}$ can be computed by a circuit of size $(nk^2 + m'k^{1-\delta/2})\,\mathsf{poly}(\lambda)$, which is sublinear in $m' \cdot k$ when k is set appropriately. We break down the task of computing $\mathsf{SI}_1, \ldots, \mathsf{SI}_{m'}$ in two steps.

1. Clearly, to compute each SI_i in amortized sublinear time in k, we cannot afford to compute the entire row Corr_i which has dimension k. Instead, we compute the list NZCorr_i of non-zero entries in Corr_i only, which has size $O(k^{1-\delta})$. More precisely, NZCorr_i consists of tuples of the form
$$\mathsf{NZCorr}_i = \{(j,\ \boldsymbol{\phi}(j) = (j_1, j_2, j_3),\ \mathsf{Corr}[i,j]) \mid j \in [k],\ \mathsf{Corr}_i[j] \neq 0\}\ .$$
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 SI_i.

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

The second step: Given NZCorr_i, we can compute SI_i in time $\mathsf{poly}(\lambda)(k^{1-\delta/2})$. 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 $(j, \boldsymbol{\phi}(j) = (j_1, j_2, j_3), \mathsf{Corr}[i,j])$ 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 $\mathbf{M}_{i,\gamma} = \mathbf{U}_{i,\gamma}\mathbf{V}_{i,\gamma}^{\top}$.

The first step: Next, we first illustrate how to generate all lists $\{\mathsf{NZCorr}_i\}_{i\in[m']}$ in sublinear time in $m'k$, 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 $\{\boldsymbol{e}_j\}_{j\in[k]}$ used to encrypt the inputs $\{\boldsymbol{x}_j\}_{j\in[k]}$. More precisely, for every coordinate $l \in [n]$ in an input, form the list
$$\mathsf{NZInp}_l = \{(j, x_{j,l}, e_{j,l}) \mid e_{j,l} \neq 0\}_{j\in[k]}\ .$$
That is, NZInp_l contains the index j of each input $\boldsymbol{x}_j$, 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 $O(nk)$.

Now, think of a database that contains all $\{\mathsf{NZInp}_l\}_l$ and inputs $\{\boldsymbol{x}_j\}_j$, which can be randomly accessed. The second sub-step makes a pass over all monomials $Q_1, \ldots Q_{m'}$. Each monomial Q_i depends on at most d variables (out of n 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 $O(k^{1-\delta})$. Thus, in the RAM model $\{\mathsf{NZCorr}_i\}_i$ can be constructed in sublinear time $O(m'k^{1-\delta})$. All we need to do now is coming up with a circuit to do the same.

Circuit Conversion. 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 Section 4.2.2 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 $\{i_1, \dots, i_q\}$ and a database $\mathsf{DB} \in \{0,1\}^N$ and it outputs $\{\mathsf{DB}[i_1], \dots, \mathsf{DB}[i_q]\}$. We show that this can be implemented efficiently by a circuit:

Lemma 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 4.2.2 for how we use the above lemma and other technical details.

2.2.1 Outline

This completes are technical overview. In the rest of the paper, we formally define and construct Functional Encryption in Section 7. In Section 7.2 we define the notion of a Partially Hiding Functional Encryption scheme. In Section 6, we define and construct the notion of our PRE scheme. Finally in Section 7 we show how to construct a sublinear functional encryption and $i\mathcal{O}$ from these two primitives. PRE relies on two building blocks ARE and PPE in itself. They are defined and constructed in Section 5 and Section 4 respectively. The outline is summarized in Figure 1.

3 Preliminaries

We now set up some notations that will be used throughout the paper. 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 $\boldsymbol{b}$ and $\boldsymbol{u}$. Matrices will be denoted by capitalized bold-faced letters for such as $\boldsymbol{A}$ and $\boldsymbol{M}$. For any $k \in \mathbb{N}$, we denote by the notation $\boldsymbol{v}^{\otimes k} = \underbrace{\boldsymbol{v} \otimes \cdots \otimes \boldsymbol{v}}_{k}$ the standard tensor product. This contains all the monomials in the variables inside $\boldsymbol{v}$ 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 $f \in \mathbb{Z}[\boldsymbol{x} = (x_1, \dots, x_n)]$, mapping $\{0,1\}^n$ to $\{0,1\}$. At times, we consider a mapping of such polynomial $f \in \mathbb{Z}[\boldsymbol{x}]$ into a polynomial g over $\mathbb{Z}_p[\boldsymbol{x}]$ 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 $g(\boldsymbol{x}) = f(\boldsymbol{x}) \mod p$ for every

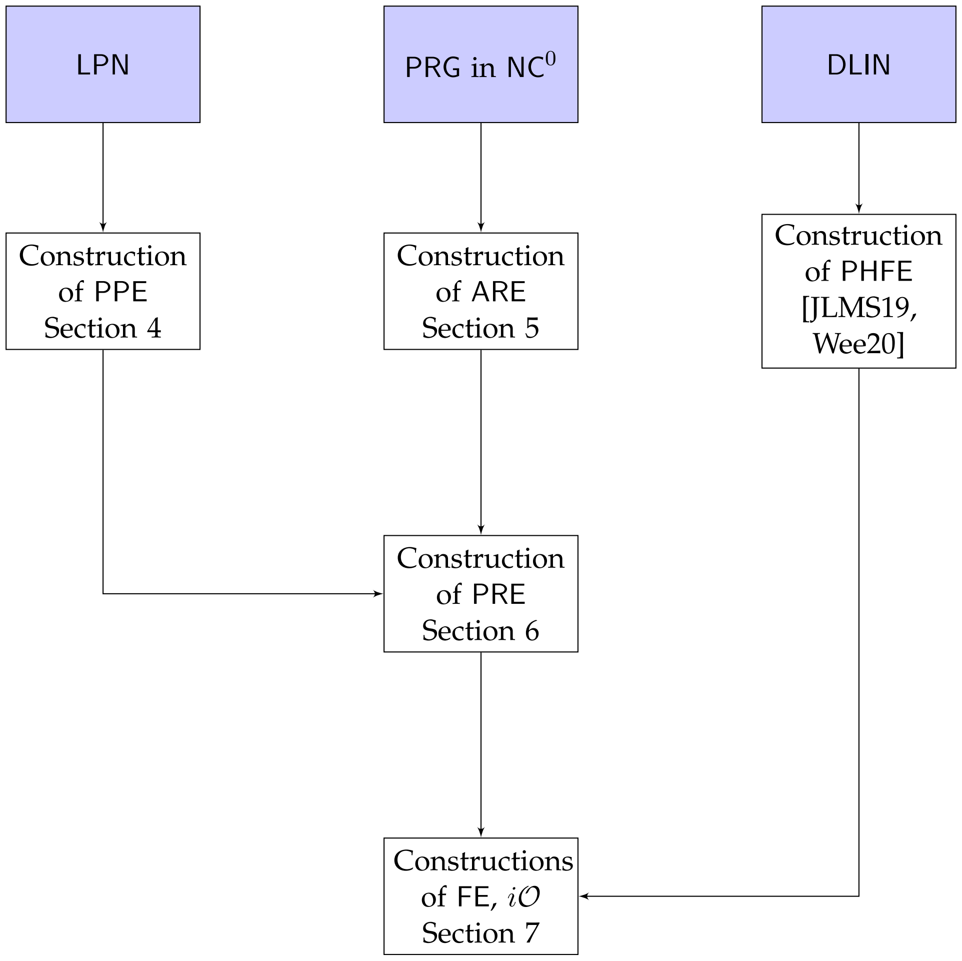
LPN
PRG in NC^0
DLIN
Construction of PPE Section 4
Construction of ARE Section 5
Construction of PHFE [JLMS19, Wee20]
Construction of PRE Section 6
Constructions of FE, $i\mathcal{O}$ Section 7

Figure 1: Flowchart depicting the technical outline.

$\boldsymbol{x} \in \{0,1\}^n$ as $f(\boldsymbol{x}) \in \{0,1\}$ for every such $\boldsymbol{x}$. Furthermore, given any NC^0 function F, finding these representations take polynomial time.

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

Definition 3.1 (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.

Assumptions We make use of three assumptions. We state the two assumptions LPN and PRG below, which are used to build the components which are new to this paper. Please see [JLS21] for a formal definition of DLIN.

Definition 3.2 (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:*

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

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

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

The second assumption we use is of that of an existence of Boolean PRG in NC^0 with polynomial stretch.

Definition 3.3. *(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 3.1. 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].

4 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 4.1 (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 $\boldsymbol{x} \in \{0,1\}^n$ and a set $Q \subseteq [n]$, define $\mathsf{Mon}_Q(\boldsymbol{x}) = \prod_{i \in Q} x_i$ to be the monomial in $\boldsymbol{x}$ corresponding to the set Q. Thus, for any input $\boldsymbol{x}$, 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 4.2 (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 $\boldsymbol{x}$ consisting of k_{PPE} blocks $\boldsymbol{x} = (\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(k_{\mathsf{PPE}})})$ each of n_{PPE} variables, and has form:*

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

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 $(\mathsf{PI}, \mathsf{SI})$ will be sublinear in $|\mathcal{Q}| \cdot k_{\mathsf{PPE}}$.

Definition 4.3 (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:*

- $(\mathsf{PI}, \mathsf{SI}) \leftarrow \mathsf{PreProc}(1^{n_{\mathsf{PPE}}},\ 1^{k_{\mathsf{PPE}}},\ p,\ \mathcal{Q}, \boldsymbol{x} \in \mathbb{Z}_p^{n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}})$: *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 $\boldsymbol{x} \in \mathbb{Z}_p^{n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}}$. It processes it to output two strings, a public string* PI *and a private string* SI*. 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* $(\mathsf{PI}, \mathsf{SI})$ *over $\mathbb{Z}_p$.*

- $y \leftarrow \mathsf{Eval}(f \in \mathcal{F}_{\mathsf{PPE},d,n_{\mathsf{PPE}},\mathcal{Q},k_{\mathsf{PPE}}}, (\mathsf{PI}, \mathsf{SI}))$: *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* $(\mathsf{PI}, \mathsf{SI})$*. It outputs $y \in \mathbb{Z}_p$.*

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

Definition 4.4 ((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 $\boldsymbol{x} \in \mathbb{Z}_p^{n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}}$:*

$$\Pr\Big[\ \mathsf{Eval}(f,\ (\mathsf{PI}, \mathsf{SI})) = f(\boldsymbol{x}) \bmod p\ \Big|\ (\mathsf{PI}, \mathsf{SI}) \leftarrow \mathsf{PreProc}(1^{n_{\mathsf{PPE}}}, 1^{k_{\mathsf{PPE}}}, p, \mathcal{Q}, \boldsymbol{x})]\ \Big] \geq 1 - O(\exp(-{k_{\mathsf{PPE}}}^{\Omega(1)}))$$

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 $\boldsymbol{x} \in \mathbb{Z}_p^{n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}}$, the public part of the computed pre-processed input while pre-processing $\boldsymbol{x}$ 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 4.5 (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 $\{\boldsymbol{x}_{k_{\mathsf{PPE}}}\}_{k_{\mathsf{PPE}} \in \mathbb{N}}$ be any ensemble of inputs where each $\boldsymbol{x}_{k_{\mathsf{PPE}}} \in \mathbb{Z}_p^{n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}}$ 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* $\mathsf{negl}(k_{\mathsf{PPE}})$.

$$\left\{ \mathsf{PI} \mid (\mathsf{PI},\ \mathsf{SI}) \leftarrow \mathsf{PreProc}(1^{n_{\mathsf{PPE}}},\ 1^{k_{\mathsf{PPE}}},\ p,\ \mathcal{Q}_{k_{\mathsf{PPE}}},\ \boldsymbol{x}_{k_{\mathsf{PPE}}}) \right\}_{k_{\mathsf{PPE}}}$$

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

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

Definition 4.6 (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 $t_{\mathsf{PPE}} = \widetilde{O}((m_{\mathsf{PPE}} k_{\mathsf{PPE}})^{1-\Omega(1)})$ where $\widetilde{O}$ 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 SI.

Definition 4.7 (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 $\boldsymbol{x} \in \mathbb{Z}_p^{n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}}$, any pre-processed input* $(\mathsf{PI},\mathsf{SI}) \leftarrow \mathsf{PreProc}(1^{n_{\mathsf{PPE}}},\ 1^{k_{\mathsf{PPE}}},\ p,\ \Gamma,\ \boldsymbol{x})$*, and any $f \in \mathcal{F}_{d,n_{\mathsf{PPE}},\mathcal{Q},k_{\mathsf{PPE}}}$, the following relation is satisfied:*

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

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

4.1 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

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

- Parse $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{k_{\mathsf{PPE}}})$ where each $\boldsymbol{x}_j \in \mathbb{Z}_p^{n_{\mathsf{PPE}}}$. Parse $\boldsymbol{x}_j = (x_{j,1}, \ldots, x_{j,n_{\mathsf{PPE}}})$.
- The overall outline is the following: We first show how to sample components $\mathsf{PI}' = (\mathsf{PI}_1, \ldots, \mathsf{PI}_{k_{\mathsf{PPE}}})$, and then how to sample SI along with a boolean variable flag. PI will be set as $(\mathsf{flag}, \mathsf{PI}')$.
- **Sampling** $\mathsf{PI}' = (\mathsf{PI}_1, \ldots, \mathsf{PI}_{k_{\mathsf{PPE}}})$**:** Sample $\boldsymbol{s} \leftarrow \mathbb{Z}_p^{k_{\mathsf{PPE}}}$. For every $i \in [n_{\mathsf{PPE}}]$, and $j \in [k_{\mathsf{PPE}}]$:
  1. Sample $\boldsymbol{a}_{j,i} \leftarrow \mathbb{Z}_p^{k_{\mathsf{PPE}}}$.
  2. Sample $e_{j,i} \leftarrow \mathsf{Ber}({k_{\mathsf{PPE}}}^{-\delta}) \cdot \mathbb{Z}_p$. Denote $\boldsymbol{e}_j = (e_{j,1}, \ldots, e_{j,n_{\mathsf{PPE}}})$.
  3. Compute $b_{j,i} = \langle \boldsymbol{a}_{j,i}, \boldsymbol{s} \rangle + e_{j,i} + x_{j,i} \mod p$.

  For $j \in [k_{\mathsf{PPE}}]$, set $\mathsf{PI}_j = \{\boldsymbol{a}_{j,i}, b_{j,i}\}_{i \in [n_{\mathsf{PPE}}]}$.
- **Sampling** SI**:** SI has $m_{\mathsf{PPE}} + 1$ components. That is, $\mathsf{SI} = (\mathsf{SI}_0, \ldots, \mathsf{SI}_{m_{\mathsf{PPE}}})$. Set $\mathsf{SI}_0 = (1, \boldsymbol{s})^{\otimes \lceil \frac{d}{2} \rceil}$. We now show how to compute SI_r for $r \in [m_{\mathsf{PPE}}]$.
  1. For $j \in [k_{\mathsf{PPE}}]$, compute $\mathsf{Corr}_{r,j} = \mathsf{Mon}_{Q_r}(\boldsymbol{x}_j) - \mathsf{Mon}_{Q_r}(\boldsymbol{x}_j + \boldsymbol{e}_j)$.
  2. Initialize for every $\gamma \in [t_1]$, matrices $\boldsymbol{M}_{r,\gamma}$ 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 $\boldsymbol{M}_{r,j_1}[j_2, j_3] = \mathsf{Corr}_{r,j}$. If any matrix $\boldsymbol{M}_{r,\gamma}$ for $\gamma \in [t_1]$, has more than t 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 $\boldsymbol{U}_{r,\gamma}, \boldsymbol{V}_{r,\gamma}^{\top} \in \mathbb{Z}_p^{T \times t}$ such that $\boldsymbol{M}_{r,\gamma} = \boldsymbol{U}_{r,\gamma} \cdot \boldsymbol{V}_{r,\gamma}$. Otherwise for every $\gamma \in [t_1]$, set $\boldsymbol{U}_{r,\gamma}, \boldsymbol{V}_{r,\gamma}$ to be matrices with zero-entries.
  5. Set $\mathsf{SI}_r = \{\boldsymbol{U}_{r,\gamma}, \boldsymbol{V}_{r,\gamma}\}_{\gamma \in [t_1]}$.
- **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 $[{k_{\mathsf{PPE}}}^{1-\delta} - t{k_{\mathsf{PPE}}}^{\frac{1-\delta}{2}}, {k_{\mathsf{PPE}}}^{1-\delta} + t{k_{\mathsf{PPE}}}^{\frac{1-\delta}{2}}]$, set $\mathsf{flag} = 0$. Otherwise, set $\mathsf{flag} = \min\{\mathsf{flag}_r\}_{r \in [m]}$.

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

$\mathbb{Z}$. Output:

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

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

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

where $\phi(j) = (j_1,(j_2,j_3))$. We remark that the polynomial above is written as a function of $\boldsymbol{s}$ and not SI_0, however, since we always mean $\mathsf{SI}_0 = (1,\boldsymbol{s})^{\otimes\lceil\frac{d}{2}\rceil}$, we treat this polynomial as some degree-2 polynomial in SI_0.

Remark 4.1. 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 $(\mathsf{PI},\mathsf{SI})$ 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 $\mathsf{Eval}(f,(\mathsf{PI},\mathsf{SI}))$ 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 SI, and $O(d)$ in PI. 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 SI as it is degree 2 in matrices $\{\boldsymbol{U}_{r,\gamma},\boldsymbol{V}_{r,\gamma}\}_{\gamma\in[t_1]}$. The degree of the second term in PI is one, as it is degree one in flag.
- The first term is at most degree $d+1$ in PI as it is degree one in flag and at most degree d in $\boldsymbol{a}_{j,i}$ and $b_{j,i}$ variables. It is also at most degree d in $\boldsymbol{s}$, however we interpret $\mathsf{SI}_0 = (1,\boldsymbol{s})^{\otimes\lceil\frac{d}{2}\rceil}$. Therefore, when written as a polynomial in SI_0, its degree is 2.

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

Lemma 4.1. *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 $\boldsymbol{x} \in \mathbb{Z}_p^{n_{\mathsf{PPE}}\cdot k_{\mathsf{PPE}}}$, when computing $(\mathsf{PI},\mathsf{SI}) \leftarrow \mathsf{PreProc}(1^{n_{\mathsf{PPE}}},1^{k_{\mathsf{PPE}}},p,\mathcal{Q},\boldsymbol{x})$:*

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

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 SI.

Claim 4.1. *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, $\boldsymbol{M}_{r,\gamma}$ for some $\gamma \in [t_1]$ has more than t non zero entries. The number of entries in $\boldsymbol{M}_{r,\gamma}$ 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 $\boldsymbol{M}_{r,\gamma}$ for any given $\gamma \in [t_1]$ has more than t non-zero entries is at most:

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

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

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

Claim 4.2. *For any $i \in [n_{\mathsf{PPE}}]$, $\Pr\left[|\mathsf{Set}_i| \notin [{k_{\mathsf{PPE}}}^{1-\delta} - t \cdot {k_{\mathsf{PPE}}}^{\frac{1-\delta}{2}}, {k_{\mathsf{PPE}}}^{1-\delta} + t \cdot {k_{\mathsf{PPE}}}^{\frac{1-\delta}{2}}]\right] = O(\exp(-{k_{\mathsf{PPE}}}^{\Omega(1)}))$.*

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 $\rho = \frac{t}{{k_{\mathsf{PPE}}}^{\frac{1-\delta}{2}}}$ and plugging into the bound gives us the required probability. This comes up to $2\exp(-\frac{t^2}{2})$, which is $O(\exp(-{k_{\mathsf{PPE}}}^{\Omega(1)}))$. □

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

Security: We now argue security.

Lemma 4.2. *If δ-LPN holds, then, the PPE scheme described above is secure as per Definition 4.3. 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 PI. Then, we show indistinguishable hybrids and argue indistinguishability between them. The first hybrid corresponds to the case when PI is generated by preprocessing $\boldsymbol{x} \in \mathbb{Z}_p^{n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}}$, and the last hybrid is independent of $\boldsymbol{x}$. Recall,

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

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

$\mathbf{Hybrid}_1$: 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 4.1, $\mathbf{Hybrid}_0$ 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 $\{\boldsymbol{a}_{j,i}, b_{j,i}\}_{j\in[k_{\mathsf{PPE}}],i\in[n_{\mathsf{PPE}}]}$ is generated. In $\mathbf{Hybrid}_1$, $\boldsymbol{a}_{j,i}$ is generated as a random vector sampled from $\mathbb{Z}_p^{k_{\mathsf{PPE}}}$. $\{b_{j,i} = \langle \boldsymbol{a}_{j,i}, \boldsymbol{s}\rangle + e_{j,i} + x_{j,i} \mod p\}_{j\in[k_{\mathsf{PPE}}],i\in[n_{\mathsf{PPE}}]}$, where $\boldsymbol{s}$ 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. □

Correctness: We now argue correctness:

Lemma 4.3. *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, $\boldsymbol{x} \in \mathbb{Z}_p^{n_{\mathsf{PPE}}\cdot k_{\mathsf{PPE}}}$. Let $(\mathsf{PI}, \mathsf{SI}) \leftarrow \mathsf{PreProc}(1^{n_{\mathsf{PPE}}}, 1^{k_{\mathsf{PPE}}}, p, \mathcal{Q}, \boldsymbol{x})$, then, for any $f \in \mathcal{F}_{\mathsf{PPE},d,n_{\mathsf{PPE}},\mathcal{Q},k_{\mathsf{PPE}}}$:*

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

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

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

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

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

where $\phi(j) = (j_1, (j_2, j_3))$. As shown in Lemma 4.1, $\mathsf{flag} = 1$ with probability $1-O(\exp(-{k_{\mathsf{PPE}}}^{\Omega(1)}))$. Thus, with this probability,

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

Also observe that:

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

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

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

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

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

This completes the proof. □

4.2 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 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 4.4 (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 4.5. *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 4.8 (RAM lookup program). *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 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 4.6. *Let $q, N \in \mathbb{N}$. A RAM lookup program $P^{\mathsf{RAM}}_{q,N}$ 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.*

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 $\boldsymbol{y} = (y_1, \ldots, y_q) \in [N]^q$, compute the following set $\boldsymbol{z}_1 = \{(1, y_1), \ldots, (q, y_q)\}$ that contains q tuples. Namely, append each query with its order.

Step 2: Compute $\boldsymbol{z}_2$ which is obtained by sorting the tuples inside $\boldsymbol{z}_1$ 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 $\boldsymbol{z}_1$ where the tuples are arranged in increasing order of the query locations in $[N]$.

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

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

Step 5: Finally, process $\boldsymbol{z}_4$ and remove the first component of the tuple to produce the output $\boldsymbol{z}_5 = (\mathsf{DB}[y_1], \ldots, \mathsf{DB}[y_q])$. 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 $\boldsymbol{y}$ and $\boldsymbol{z}_4$ respectively. Thus from Lemma 4.5 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 4.4, 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 $\boldsymbol{z}_2$.

We load on the first tape DB and on the second tape, $\boldsymbol{z}_2$. The third tape will be used to write the output $\boldsymbol{z}_3$. 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 $\boldsymbol{z}_2$, 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 $\boldsymbol{z}_2$ 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 4.5, 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. $\square$

4.2.2 Sublinear Preprocessing Efficiency

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

Theorem 4.1. *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 $\widetilde{O}({k_{\mathsf{PPE}}}^{\lceil \frac{d}{2} \rceil} + n_{\mathsf{PPE}} \cdot {k_{\mathsf{PPE}}}^2 + m_{\mathsf{PPE}} \cdot {k_{\mathsf{PPE}}}^{1 - \frac{2\delta}{5}})$ where $\widetilde{O}$ 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 $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{k_{\mathsf{PPE}}}) \in \mathbb{Z}_p^{n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}}$, $\mathsf{PreProc}(1^{n_{\mathsf{PPE}}}, 1^{k_{\mathsf{PPE}}}, p, \mathcal{Q}, \cdot)$ produces the following components:

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

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

1. Random vectors $\{\boldsymbol{a}_{j,i}\}_{j\in[k_{\mathsf{PPE}}],i\in[n_{\mathsf{PPE}}]}$ and the secret $\boldsymbol{s}$ 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 $(\mathsf{PI}_1, \ldots, \mathsf{PI}_{k_{\mathsf{PPE}}})$, the second one computes SI_0, and the third one computes $(\mathsf{flag}, \mathsf{SI}_1, \ldots, \mathsf{SI}_{m_{\mathsf{PPE}}})$. 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 $(\mathsf{PI}_1, \ldots, \mathsf{PI}_{k_{\mathsf{PPE}}})$: Observe that PI_j is simply $\{\boldsymbol{a}_{j,i}, \langle \boldsymbol{a}_{j,i}, \boldsymbol{s}\rangle + e_{j,i} + x_{j,i} \mod p\}_{i\in[n_{\mathsf{PPE}}]}$. 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 SI_0: Note that SI_0 consists simply of $(1, \boldsymbol{s})^{\otimes\lceil \frac{d}{2}\rceil}$. 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 $(\mathsf{flag}, \mathsf{SI}_1, \ldots, \mathsf{SI}_{m_{\mathsf{PPE}}})$: In Figure 2 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 $\boldsymbol{\alpha}_0$ 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 $\boldsymbol{\alpha}_1$ 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 4.4, 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.
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 $[{k_{\mathsf{PPE}}}^{1-\delta} - t \cdot {k_{\mathsf{PPE}}}^{\frac{1-\delta}{2}}, {k_{\mathsf{PPE}}}^{1-\delta} + t \cdot {k_{\mathsf{PPE}}}^{\frac{1-\delta}{2}}]$. Otherwise it is set as 1.

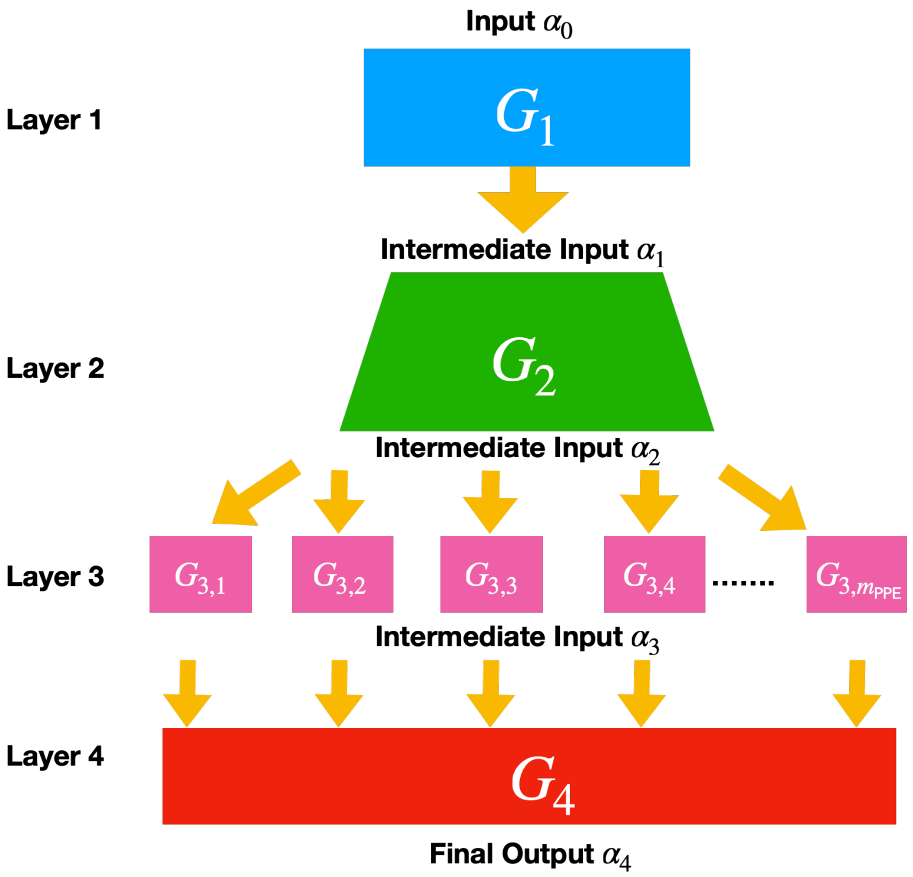

Figure 2: Circuit template for the third circuit.

2. The output $\boldsymbol{\alpha}_2$ consists of flag_{G_2} along with n_{PPE} components, $\boldsymbol{\alpha}_{2,\ell}$ for $\ell \in [n_{\mathsf{PPE}}]$. $\boldsymbol{\alpha}_{2,\ell}$ consists of at most $k_{\mathsf{PPE}}{}^{1-\delta} + t \cdot k_{\mathsf{PPE}}{}^{\frac{1-\delta}{2}}$ tuples. These are all tuples that occur in $\boldsymbol{\alpha}_1$, 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.

<u>SIZE OF G_2:</u> Note that both these steps above can be performed by a two tape Turing machine that keeps $\boldsymbol{\alpha}_1$ 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 $\{\boldsymbol{\alpha}_{2,\ell}\}_{\ell \in [n_{\mathsf{PPE}}]}$. Since $\boldsymbol{\alpha}_1$ already consists of tuples that are sorted, each $\{\boldsymbol{\alpha}_{2,\ell}\}$ 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 $\boldsymbol{\alpha}_1$. 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 4.5, 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 $(\boldsymbol{\alpha}_{3,r}, \mathsf{flag}_r \in \{0,1\})$. 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 $\{\boldsymbol{\alpha}_{2,\ell}\}_{\ell \in [Q_r]}$. Then the circuit does the following:

1. Combine the tuples $\{\boldsymbol{\alpha}_{2,\ell}\}_{\ell \in Q_r}$ 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. Otherwise set $\mathsf{flag}_r = 1$.

3. Compute $\boldsymbol{\alpha}_{3,r}$ which consists of a preprocessing of B_r, done as follows. If $\mathsf{flag}_r = 0$, set $\boldsymbol{\alpha}_{3,r} = \perp$, 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 $(\mathsf{flag}_r, \boldsymbol{\alpha}_{3,r})$. If $\mathsf{flag}_r = 0$, $\boldsymbol{\alpha}_{3,r} = \perp$. Otherwise if $\mathsf{flag}_r = 1$ (which happens with overwhelming probability), then $\boldsymbol{\alpha}_{3,r}$ 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 $(\mathsf{SI}_1, \ldots, \mathsf{SI}_{m_{\mathsf{PPE}}})$ 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 4.4 it can be implemented by a circuit in size $O(({k_{\mathsf{PPE}}}^{1-\delta} + t{k_{\mathsf{PPE}}}^{\frac{1-\delta}{2}}) \cdot \mathsf{poly}(\log_2(k_{\mathsf{PPE}} \cdot n_{\mathsf{PPE}} \cdot p))) = O({k_{\mathsf{PPE}}}^{1-\delta}\, \mathsf{poly}(\log_2(n_{\mathsf{PPE}} k_{\mathsf{PPE}} p)))$.

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 4.5 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 4.5 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}.
- $\{e_{j,i}, x_{j,i}\}_{j \in [k_{\mathsf{PPE}}], i \in [n_{\mathsf{PPE}}]}$.
- $\{\boldsymbol{\alpha}_{3,r}\}_{r \in [m_{\mathsf{PPE}}]}$.

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 $\{\boldsymbol{\beta}_r\}_{r \in [m_{\mathsf{PPE}}]}$.

- This is the output generating step. For $r \in [m_{\mathsf{PPE}}]$, one by one, read flag_r in parallel with $\boldsymbol{\beta}_r$.
  1. If $\mathsf{flag}_r = 0$, output $\mathsf{SI}_r = \{\boldsymbol{U}_{r,\gamma}, \boldsymbol{V}_{r,\gamma}^\top\}_{\gamma \in [t_1]}$ where $\boldsymbol{U}_{r,\gamma} = \boldsymbol{V}_{r,\gamma}^\top = 0^{T \times t}$.
  2. If $\mathsf{flag}_r = 1$, do the following. First let $\boldsymbol{\beta}_r = \{\boldsymbol{\beta}_{r,\gamma}\}_{\gamma \in [t_1]}$ where $\boldsymbol{\beta}_{r,\gamma}$ consists of all tuples in $\boldsymbol{\beta}_r$ 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 $\boldsymbol{\beta}_{r,\gamma}$. Note that $D_{r,\gamma} \leq t$ (as ensured by setting $\mathsf{flag}_r = 1$). For every $\gamma \in [t_1]$:
     (a) For $l \in [t]$, set vectors $\boldsymbol{u}_{r,\gamma,l} = \boldsymbol{v}_{r,\gamma,l} = 0^T$.
     (b) For $l \in D_{r,\gamma}$, parse l^{th} tuple in $\boldsymbol{\beta}_{r,\gamma}$ as $(Q_r, j, \gamma, (j_2, j_3), \{e_{j,i}, x_{j,i}\}_{i \in Q_r})$. Set $\boldsymbol{u}_{r,\gamma,l}[j_2] = \Pi_{i \in Q_r} x_{j,i} - \Pi_{i \in Q_r}(x_{j,i} + e_{j,i})$ (which is equal to $\mathsf{Corr}_{r,j}$) and $\boldsymbol{v}_{r,\gamma,l}[j_3] = 1$.
     (c) Set $\boldsymbol{U}_{r,\gamma}$ as concatenation of vectors $\{\boldsymbol{u}_{r,\gamma,l}\}_{l \in [t]}$ and $\boldsymbol{V}_{r,\gamma}^\top$ as concatenation of $\{\boldsymbol{v}_{r,\gamma,l}\}_{l \in [t]}$. Output $\mathsf{SI}_r = \{\boldsymbol{U}_{r,\gamma}, \boldsymbol{V}_{r,\gamma}^\top\}_{\gamma \in [t_1]}$.

We first argue about the size of the circuit implementing G_4 and then argue its correctness.

<u>SIZE OF G_4</u>: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 $\{\boldsymbol{\alpha}_{3,r}\}_{r \in [m_{\mathsf{PPE}}]}$, 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 4.6. The third step can be simulated by a multi-tape turing machine with a constant number of tapes in $O(m_{\mathsf{PPE}} \cdot t_1 \cdot T \cdot t\,\mathsf{poly}(\log_2 n_{\mathsf{PPE}} \cdot p \cdot k_{\mathsf{PPE}}))$ steps. And thus, by Lemma 4.5 it can be implemented by a circuit of size $O(m_{\mathsf{PPE}} \cdot t_1 \cdot T \cdot t\,\mathsf{poly}(\log_2 n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}} \cdot p))$ for some polynomial poly. Let us elucidate: For every $r \in [m_{\mathsf{PPE}}]$, the Turing machine keeps $\boldsymbol{\beta}_r$ 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 $2 \cdot t_1 \cdot t \cdot T$ field elements (which consists SI_r which consists of all 0 matrices) on a third output tape, which takes $O(t_1 \cdot t \cdot T\,\mathsf{poly}(\log_2 n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}} \cdot p))$ steps.

- In the condition where $\mathsf{flag}_r = 1$, the circuit parses the tuples in $\boldsymbol{\beta}_r$ sequentially. There are at most $O({k_{\mathsf{PPE}}}^{1-\delta}) = O(t_1)$ tuples in all that are divided into $\{\boldsymbol{\beta}_{r,\gamma}\}_{\gamma \in t_1}$. 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 $\boldsymbol{U}_{r,\gamma}, \boldsymbol{V}_{r,\gamma}$. This is done by sequentially processing tuples in $\boldsymbol{\beta}_{r,\gamma}$. For every tuple encountered (and say it is the l^{th} tuple where $l \leq D_{r,\gamma} \leq t$) 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 $\boldsymbol{u}_{r,\gamma,l}$ and $\boldsymbol{v}_{r,\gamma,l} \in \mathbb{Z}_p^T$ where $\boldsymbol{u}_{r,\gamma,l}[j_2] = \mathsf{Mon}_{Q_r}(\boldsymbol{x}_j) - \mathsf{Mon}_{Q_r}(\boldsymbol{x}_j + \boldsymbol{e}_j)$ and 0 otherwise. Likewise, $\boldsymbol{v}_{r,\gamma,l}[j_3] = 1$ and 0 otherwise. If the number of tuples is less than t, it prints $t - D_{r,\gamma}$ additional zero vectors $\{\boldsymbol{u}_{r,\gamma,l}, \boldsymbol{v}_{r,\gamma,l}\}_{l \in [t] \setminus [D_{r,\gamma}]}$. This can be done by making a single forward pass on $\boldsymbol{\beta}_{r,\gamma}$ and doing $O(T)$ arithmetic/comparision operations on it per tuple. Thus, it takes at most $O(t \cdot T\,\mathsf{poly}(\log_2(n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}} \cdot p)))$ steps to output $\{\boldsymbol{U}_{r,\gamma}, \boldsymbol{V}_{r,\gamma}^\top\}$. Thus the total time complexity to output SI_r is $O(t_1 \cdot t \cdot T\,\mathsf{poly}(\log_2(p \cdot k_{\mathsf{PPE}} \cdot n_{\mathsf{PPE}})))$ for some polynomial poly. Thus, the complexity to output $(\mathsf{SI}_1, \ldots, \mathsf{SI}_{m_{\mathsf{PPE}}})$ is $O(m_{\mathsf{PPE}} \cdot t_1 \cdot t \cdot T\,\mathsf{poly}(\log_2(n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}} \cdot p)))$ 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 $\boldsymbol{\alpha}_{3,r}$ which consists of all the indices $j \in [k_{\mathsf{PPE}}]$ for which $\mathsf{Mon}_{Q_r}(\boldsymbol{x}_j) \neq \mathsf{Mon}_{Q_r}(\boldsymbol{x}_j + \boldsymbol{e}_j)$. This is the list of indices which need correction for monomial Q_r. Then, for every tuple that $\boldsymbol{\alpha}_{3,r}$ contains, we load up all the inputs and the errors that it needs to do this correction. This updated input is $\boldsymbol{\beta}_r$. With overwhelming probability (that is when $\mathsf{flag}_r = 1$) the size of $\boldsymbol{\beta}_{r,\gamma}$ for every $\gamma \in [t_1]$ is less than or equal to t. Then, all these corrections are embedded inside the matrices $\{\boldsymbol{U}_{r,\gamma}, \boldsymbol{V}_{r,\gamma}^{\top}\}$ where every correction is embedded in one of the t distinct T dimensional columns of $\boldsymbol{U}_{r,\gamma}$ and $\boldsymbol{V}_{r,\gamma}^{\top}$. 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 $\mathsf{Mon}_{Q_r}(\boldsymbol{x}_j)$ was the l^*th monomial for the set Q_r corrected in the j_1th bucket then,

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

Because of this G_4 produces the right output.

<u>OVERALL CIRCUIT-SIZE CALCULATION:</u> Hiding the polynomial in logarithmic multiplicative factors in $n_{\mathsf{PPE}}, p, k_{\mathsf{PPE}}$ withing $\widetilde{O}(\cdot)$ we get the following:

- $\mathsf{size}_1 = \widetilde{O}(n_{\mathsf{PPE}} \cdot {k_{\mathsf{PPE}}}^2)$,
- $\mathsf{size}_2 = \widetilde{O}({k_{\mathsf{PPE}}}^{\lceil \frac{d}{2} \rceil})$,
- $\mathsf{size}_{G_1} = \widetilde{O}(n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}})$
- $\mathsf{size}_{G_2} = \widetilde{O}(n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}})$
- $\mathsf{size}_{G_3} = \widetilde{O}(m_{\mathsf{PPE}} \cdot {k_{\mathsf{PPE}}}^{1-\delta})$ (adding the size of all m_{PPE} sub-circuits)
- $\mathsf{size}_{G_4} = \widetilde{O}(m_{\mathsf{PPE}} \cdot {k_{\mathsf{PPE}}}^{1-\frac{2\delta}{5}})$.

Combining these observations, we get our result. □

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

Theorem 4.2. *Assuming δ-*LPN *assumption (Definition 3.2) holds for any constant $\delta > 0$, then there exists a* PPE *scheme satisfying Definition 4.3. Further, if the assumption is subexponentially secure, then so is the resulting* PPE *scheme.*

5 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 $C : \{0,1\}^{n_{\mathsf{ARE}}} \to \{0,1\}^{m_{\mathsf{ARE}} \cdot k_{\mathsf{ARE}}}$ where every bit of the output is computed by a circuit of size λ. Such an ARE scheme has the following syntax:

Definition 5.1 (Syntax of ARE). *An* ARE *scheme consists of the following p.p.t. algorithms:*

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

An ARE scheme satisfies the following properties.

Definition 5.2 ((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* $\boldsymbol{x} \in \{0,1\}^{n_{\mathsf{ARE}}}$, $C \in \mathcal{F}_{\mathsf{ARE}, n_{\mathsf{ARE}}, m_{\mathsf{ARE}}, k_{\mathsf{ARE}}, \lambda}$:

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

Definition 5.3 (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* $\{C, \boldsymbol{x}_0, \boldsymbol{x}_1\}_\lambda$ *where* $\boldsymbol{x}_0, \boldsymbol{x}_1 \in \{0,1\}^{n_{\mathsf{ARE}}}$ *and* $C \in \mathcal{F}_{\mathsf{ARE}, n_{\mathsf{ARE}}, m_{\mathsf{ARE}}, k_{\mathsf{ARE}}, \lambda}$ *with* $C(\boldsymbol{x}_0) = C(\boldsymbol{x}_1)$, *it holds that for* $\lambda \in \mathbb{N}$ *the following distributions are computationally indistinguishable*

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

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 5.4 (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* $\boldsymbol{x} \in \{0,1\}^{n_{\mathsf{ARE}}}$, $\mathsf{Encode}(C, \boldsymbol{x}; \boldsymbol{r}) \to \boldsymbol{y} \in \{0,1\}^T$ *satisfies the following requirements:*

- *Parse* $\boldsymbol{r} = (\boldsymbol{r}_1, \ldots, \boldsymbol{r}_{k_{\mathsf{ARE}}})$ *where each component is of equal size. Let* $\boldsymbol{a}_i = (\boldsymbol{x}, \boldsymbol{r}_i)$. *Then the length of* $\boldsymbol{a}_i \in \{0,1\}^{n'_{\mathsf{ARE}}}$ *is* $n'_{\mathsf{ARE}} = O((n_{\mathsf{ARE}} + m_{\mathsf{ARE}}^{1-c_2})\lambda^{c_1})$.
- *For* $i \in [T]$, *each* $y_i = \sum_{Q \in \mathcal{Q}, j \in [k_{\mathsf{ARE}}]} \mu_{i,Q,j} \cdot \mathsf{Mon}_Q(\boldsymbol{a}_j)$ *for efficiently samplable* $\mu_{i,Q,j} \in \mathbb{Z}$.

The first property is to ensure that $\boldsymbol{a}_i$ 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 $\boldsymbol{y}$.

5.1 Construction Details

In Figure 3, 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 $n'_{\mathsf{ARE}} = (n_{\mathsf{ARE}} + m_{\mathsf{ARE}}^{1-\epsilon})\,\mathsf{poly}(\lambda)$. 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, $U(C_i, \boldsymbol{x}) = C_i(\boldsymbol{x})$ for circuits C_i and input $\boldsymbol{x}_i$ 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.

The ARE scheme

Encode $\mathsf{Encode}(C, \boldsymbol{x}, \boldsymbol{r})$**:** 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 $\boldsymbol{r} = (\boldsymbol{r}_1, \ldots, \boldsymbol{r}_{k_{\mathsf{ARE}}})$ where $\boldsymbol{r}_i \in \{0,1\}^{n'_{\mathsf{ARE}} - n_{\mathsf{ARE}}}$. Set $\boldsymbol{a}_i = (\boldsymbol{x}, \boldsymbol{r}_i) \in \{0,1\}^{n'_{\mathsf{ARE}}}$ for $i \in [k_{\mathsf{ARE}}]$. For every $\kappa \in [k_{\mathsf{ARE}}]$, compute Π_κ as follows:

- Using G expand $\boldsymbol{r}_\kappa$ into $(\boldsymbol{\sigma}, \boldsymbol{b})$ of length $(n_{\mathsf{ARE}} + m_{\mathsf{ARE}})\,\mathsf{poly}(\lambda)$. Here $\boldsymbol{\sigma}$ will be used as labels to produce garbling of $U(C_\kappa, \boldsymbol{x})$ and $\boldsymbol{b}$ 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 $\boldsymbol{x}$) Generate input labels of $(C_\kappa, \boldsymbol{x})$. That is for every input wire $w_{ckt,i}$ for $i \in [m_{\mathsf{ARE}} \cdot \lambda]$ and $w_{inp,j}$ for $j \in [n_{\mathsf{ARE}}]$.

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

  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.

$$T_{\mathsf{gate}} = \begin{pmatrix} \mathsf{H}_0(\boldsymbol{\sigma}_{w_1,b_{w_1}}) \oplus \mathsf{H}_0(\boldsymbol{\sigma}_{w_2,b_{w_2}}) \oplus \left(\boldsymbol{\sigma}_{w_3,g(b_{w_1},b_{w_2})} || g(b_{w_1}, b_{w_2}) \oplus b_{w_3}\right) \\ \mathsf{H}_1(\boldsymbol{\sigma}_{w_1,b_{w_1}}) \oplus \mathsf{H}_0(\boldsymbol{\sigma}_{w_2,\bar{b}_{w_2}}) \oplus \left(\boldsymbol{\sigma}_{w_3,g(b_{w_1},\bar{b}_{w_2})} || g(b_{w_1}, \bar{b}_{w_2}) \oplus b_{w_3}\right) \\ \mathsf{H}_0(\boldsymbol{\sigma}_{w_1,\bar{b}_{w_1}}) \oplus \mathsf{H}_1(\boldsymbol{\sigma}_{w_2,b_{w_2}}) \oplus \left(\boldsymbol{\sigma}_{w_3,g(\bar{b}_{w_1},b_{w_2})} || g(\bar{b}_{w_1}, b_{w_2}) \oplus b_{w_3}\right) \\ \mathsf{H}_1(\boldsymbol{\sigma}_{w_1,\bar{b}_{w_1}}) \oplus \mathsf{H}_1(\boldsymbol{\sigma}_{w_2,\bar{b}_{w_2}}) \oplus \left(\boldsymbol{\sigma}_{w_3,g(\bar{b}_{w_1},\bar{b}_{w_2})} || g(\bar{b}_{w_1}, \bar{b}_{w_2}) \oplus b_{w_3}\right) \end{pmatrix} \quad (2)$$

- 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 $\boldsymbol{y}_\kappa = \mathsf{YaoDecode}(\Pi_\kappa)$ for $\kappa \in [k_{\mathsf{ARE}}]$, where $\mathsf{YaoDecode}$ is the evaluation of the garbled circuit. Output $\boldsymbol{y} = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{k_{\mathsf{ARE}}})$.

Figure 3: ARE Scheme Description

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 $\boldsymbol{x}_0, \boldsymbol{x}_1$ with $C(\boldsymbol{x}_0) = C(\boldsymbol{x}_1)$. If PRG security holds, then Π computed by encoding $\boldsymbol{x}_\beta$ for a random $\beta \in$

$\{0,1\}$ is computationally indistinguishable to an honest garbling of the computation $(U(C_1, \boldsymbol{x}_\beta), \ldots, U(C_{k_{\mathsf{ARE}}}, \boldsymbol{x}_\beta))$ using truly generated randomness. But due to the security of Yao's garbling scheme, this is indistingusihable to garbling of $(U(C_1, \boldsymbol{x}_0), \ldots, U(C_{k_{\mathsf{ARE}}}, \boldsymbol{x}_0))$ which is independent of β.

Efficiency: The efficiency properties have already been argued in the overview. The size of randomness $\boldsymbol{r}_i$ is $n'_{\mathsf{ARE}} - n_{\mathsf{ARE}} = O((n_{\mathsf{ARE}} + m_{\mathsf{ARE}}^{1-\epsilon})\,\mathsf{poly}(\lambda))$. The computation of $\mathsf{Encode}(\cdot, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_{k_{\mathsf{ARE}}})$ 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 $\boldsymbol{a}_\kappa$ 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, except one i.e. the labels corresponding to the circuit input C_κ. Here too, the labels can be computed as:

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

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

Theorem 5.1. *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 3.3), then there exists an* ARE *scheme satisfying Definition 5.1. Further, if the* PRG *is subexponentially secure, then so is* ARE.

6 Preprocessed 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 6.1 (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:*

- $\mathsf{PRE.PreProc}(1^\lambda, 1^{n_{\mathsf{PRE}}}, 1^{m_{\mathsf{PRE}}}, 1^{k_{\mathsf{PRE}}}, p, \boldsymbol{x} \in \{0,1\}^{n_{\mathsf{PRE}}}) \to (\mathsf{PI}, \mathsf{SI})$. *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* $\boldsymbol{x} \in \{0,1\}^n$*. 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.*

- $\mathsf{PRE.Encode}(C, (\mathsf{PI}, \mathsf{SI})) = \boldsymbol{y}$. *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* $\boldsymbol{y}$*.*

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

Remark 6.1. 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 paper. 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 paper, 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 magnitude $\lambda^{\Theta(1)}$. 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 6.2 (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,* $\boldsymbol{x} \in \{0,1\}^{n_{\mathsf{PRE}}}$, *and* $C \in \mathcal{F}_{\mathsf{PRE}, n_{\mathsf{PRE}}, m_{\mathsf{PRE}}, k_{\mathsf{PRE}}, \lambda}$.

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

Definition 6.3 (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* $\{C, \boldsymbol{x}_0, \boldsymbol{x}_1\}_{\lambda \in \mathbb{N}}$ *be any ensemble where* $\boldsymbol{x}_0, \boldsymbol{x}_1 \in \{0,1\}^{n_{\mathsf{PRE}}(\lambda)}$ *and* $C \in \mathcal{F}_{\mathsf{PRE}, n_{\mathsf{PRE}}(\lambda), m_{\mathsf{PRE}}(\lambda), k_{\mathsf{PRE}}(\lambda), \lambda}$ *with* $\boldsymbol{y} = C(\boldsymbol{x}_0) = C(\boldsymbol{x}_1)$. *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*

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

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

The Efficiency and Complexity Requirements

Definition 6.4 (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* $\boldsymbol{x} \in \{0,1\}^{n_{\mathsf{PRE}}}$ *computes* $\mathsf{PRE.PreProc}(1^\lambda, 1^{n_{\mathsf{PRE}}},\ 1^{m_{\mathsf{PRE}}}, 1$ *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 6.5 (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* $\boldsymbol{x} \in \{0,1\}^{n_{\mathsf{PRE}}}$, *and every* $(\mathsf{PI}, \mathsf{SI}) \leftarrow \mathsf{PreProc}(1^\lambda, 1^{n_{\mathsf{PRE}}}, 1^{m_{\mathsf{PRE}}},\ 1^{k_{\mathsf{PRE}}},\ p,\ \boldsymbol{x})$,
$$f(\mathsf{PI}, \mathsf{SI}) \bmod p = \mathsf{PRE.Encode}(C, (\mathsf{PI}, \mathsf{SI}))\ .$$
- *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* $\lambda, n_{\mathsf{PRE}}, m_{\mathsf{PRE}}, k_{\mathsf{PRE}}, p, C$.

6.1 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 4:

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 $\boldsymbol{a} = (\boldsymbol{a}_1, \ldots, \boldsymbol{a}_{k_{\mathsf{ARE}}})$ where $\boldsymbol{a}_i = (\boldsymbol{x}, \boldsymbol{r}_i)$ to compute PI, SI. Then, when one runs $\mathsf{PRE.Encode}(C, (\mathsf{PI}, \mathsf{SI}))$, with overwhelming probability the output consists of $y_i = f_i(\boldsymbol{a})$ for $i \in [T]$, due to correctness of PPE scheme. This is same as $\mathsf{ARE.Encode}(C, \boldsymbol{x}, \boldsymbol{r}_1, \ldots, \boldsymbol{r}_{k_{\mathsf{ARE}}})$ by the correctness of the ARE scheme. Finally $\mathsf{PRE.Decode}(\boldsymbol{y}) = \mathsf{ARE.Decode}(\boldsymbol{y})$ which is equal to $C(\boldsymbol{x})$.

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

The PRE scheme

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

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

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 $\boldsymbol{y} = (y_1, \ldots, y_T)$.

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

Figure 4: The Description of the PRE scheme.

$\mathbf{Hybrid}_0$: In this hybrid, we compute $(\mathsf{PI}, \mathsf{SI})$ by preprocessing $\boldsymbol{a} = (\boldsymbol{a}_1, \ldots, \boldsymbol{a}_{k_{\mathsf{PRE}}})$. Here each $\boldsymbol{a}_i = (\boldsymbol{x}_b, \boldsymbol{r}_i)$ where $\boldsymbol{b} \leftarrow \{0,1\}$.

Let $\boldsymbol{y}$ be the output of $\mathsf{PRE.Encode}$ operation on input $C, \mathsf{PI}, \mathsf{SI}$. Output of the hybrid is $(\mathsf{PI}, \boldsymbol{y})$.

$\mathbf{Hybrid}_1$: In this hybrid, we compute $(\mathsf{PI}, \mathsf{SI})$ by preprocessing $\boldsymbol{a}$ generated as in the previous hybrid. Let $\boldsymbol{y}$ be the output of $\mathsf{ARE.Encode}(C, \boldsymbol{x}, \boldsymbol{r})$, where $\boldsymbol{a}_i = (\boldsymbol{x}_b, \boldsymbol{r}_i)$. Output of the hybrid is $(\mathsf{PI}, \boldsymbol{y})$.
Note that $\mathbf{Hybrid}_0$ is statistically close to $\mathbf{Hybrid}_1$ due to the correctness of PPE scheme. In one case $\boldsymbol{y}$ is actually an output of $\mathsf{PPE.Eval}$ function performed over $(\mathsf{PI}, \mathsf{SI})$ using function f_i, in the other case, it is generated by computing $y_i = f_i(\boldsymbol{a})$. The claim thus follows, due to the correctness of PPE scheme.

$\mathbf{Hybrid}_2$: In this hybrid, we compute $(\mathsf{PI}, \mathsf{SI})$ by preprocessing the all 0 string. $\boldsymbol{y}$ is computed as in the previous hybrid. Output of the hybrid is $(\mathsf{PI}, \boldsymbol{y})$.
The above two hybrids are indistinguishable due to the security of the PPE scheme. In one case PI is generated by preprocessing $\boldsymbol{a}$, in the other case it is generated by preprocessing the all 0 string.

$\mathbf{Hybrid}_3$: In this hybrid, we compute PI as in the previous hybrid. The only change is that $\boldsymbol{y}$ is computed as $\mathsf{ARE.Encode}(C, \boldsymbol{x}_0, \boldsymbol{r})$. Output of the hybrid is $(\mathsf{PI}, \boldsymbol{y})$.
The above two hybrids are indistinguishable due to the security of the ARE scheme. In both the hybrids, PI is generated by encoding 0 string. In one case $\boldsymbol{y}$ is generated by running $\boldsymbol{y} = \mathsf{ARE.Encode}(C, \boldsymbol{x}_b; \boldsymbol{r})$, and in the other, $\boldsymbol{y} = \mathsf{ARE.Encode}(C, \boldsymbol{x}_0; \boldsymbol{r})$. 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 $\mathsf{PRE.PreProc}(1^\lambda, 1^{n_{\mathsf{PRE}}}, 1^{m_{\mathsf{PRE}}}, 1^{k_{\mathsf{PRE}}}, p, \boldsymbol{x})$. This algorithm runs in two steps.

1. In the first step it arranges $\boldsymbol{x}$ and a random vector $\boldsymbol{r}$ into $\boldsymbol{a}$,
2. In the second step, it runs $\mathsf{PPE.PreProc}(1^{n_{\mathsf{PPE}}}, 1^{k_{\mathsf{PPE}}}, p, \mathcal{Q}, \boldsymbol{a})$.

The first step can be implemented by a circuit of size $O(|\boldsymbol{a}|) = O(n_{\mathsf{PRE}} \cdot k_{\mathsf{PRE}}) = O(n'_{\mathsf{ARE}} \cdot k_{\mathsf{PRE}}) = O(n_{\mathsf{PRE}} \cdot k_{\mathsf{PRE}} \cdot \lambda^{c_1} + {m_{\mathsf{PRE}}}^{1-c_2} \cdot k_{\mathsf{PRE}} \cdot \lambda^{c_1})$. 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 $\boldsymbol{y} = (g_{f_1}(\mathsf{PI}, \mathsf{SI}), \ldots, g_{f_T}(\mathsf{PI}, \mathsf{SI}))$ where $f_i(\boldsymbol{a})$ computes the i^{th} bit of $\mathsf{ARE.Encode}(C, \boldsymbol{x}, \boldsymbol{r})$. Note that due to the efficiency property of ARE f_i is of the form:

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

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 6.1. *Assuming the existence of a* PPE *scheme (Definition 4.3) and an* ARE *scheme (Definition 5.1), then the scheme above is a* PRE *scheme satisfying Definition 6.1. Further, if both the underlying primitives are subexponentially secure, then so is the resulting* PRE *scheme.*

In Theorem 4.2 it is shown that an ARE scheme satisfying Definition 5.1 can be constructed assuming PRG in NC^0 with polynomial stretch (Definition 3.3). It is shown in Theorem 4.2 that a PPE scheme satisfying Definition 4.3 can be constructed assuming the δ-LPN assumption (Definition 3.2) for any constant $\delta > 0$. Combining these two theorems we have:

Theorem 6.2. *Assume that there exists two constant $\delta, \epsilon > 0$ such that:*

- *δ-LPN assumption (Definition 3.2) holds,*
- *There exists a PRG in NC^0 with a stretch $n^{1+\epsilon}$ where n is the length of the input (Definition 3.3),*

Then, there exists a PRE *scheme (Definition 6.1). Further, assuming the underlying assumptions are subexponentially secure, then so is the resulting* PRE *scheme.*

7 Functional Encryption: Definition and Construction

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 7.1. *(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* $(\mathsf{PK}, \mathsf{MSK})$*.*
- $\mathsf{Enc}(\mathsf{PK}, \boldsymbol{x})$*: Given as input the public key* PK *and a message* $\boldsymbol{x} \in \{0,1\}^{n_{\mathsf{FE}}(\lambda)}$*, 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* $\boldsymbol{y}$ *in* $\{0,1\}^{m_{\mathsf{FE}}(\lambda)}$*, or* $\perp$ *if it fails.*

Remark 7.1 (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 paper.

We now define the correctness of decryption property.

Definition 7.2. *(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* $\boldsymbol{x} \in \{0,1\}^{n_{\mathsf{FE}}(\lambda)}$*, and every function* $f \in \mathcal{F}_{\mathsf{FE},n_{\mathsf{FE}},m_{\mathsf{FE}},\lambda}$ *we have:*

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

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

Definition 7.3 (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.*

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

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

Remark 7.2 (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 7.4 (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 7.3 (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 [AJS15a] 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.

7.1 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 7.1 ([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 [AJS15a] that, a sublinear functional encryption for this class implies sublinear functional encryption for circuits. Namely:

Theorem 7.2 ([AJS15a])**.** *Assuming there exists a public key sublinear functional encryption for $\{\mathcal{F}_{\mathsf{FE},n_{\mathsf{FE}},n_{\mathsf{FE}}^{1+\epsilon},\lambda}\}_{n_{\mathsf{FE}}\in\mathsf{poly},\lambda\in}$ 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 7.3 ([AJ15, BV15, AJS15a])**.** *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.*

7.2 Ingredient: 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 PI and a secret input SI. Any decryptor that has a function key for a function f, can learn PI along with the value $f(\mathsf{PI}, \mathsf{SI})$. Therefore a PHFE scheme for general circuits also implies a functional encryption scheme for circuits by simply setting PI 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 $\mathsf{PI}, \mathsf{SI} \in \mathbb{Z}_p^{n_{\mathsf{PHFE}}}$ and has the following form:

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

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

Definition 7.5. *(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.
- $\mathsf{Enc}(\mathsf{PK}, (\mathsf{PI}, \mathsf{SI}) \in \mathbb{Z}_p^{n_{\mathsf{PHFE}}} \times \mathbb{Z}_p^{n_{\mathsf{PHFE}}})$*: The encryption algorithm is a randomized algorithm that takes in the public key* PK *and a message* $(\mathsf{PI}, \mathsf{SI})$ *and returns the ciphertext* CT. PI *is considered as the public input and* SI *as the secret input.* CT *is implicitly assumed to have* PI *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 7.6. *(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 $(\mathsf{PI}, \mathsf{SI}) \in \mathbb{Z}_p^{n_{\mathsf{PHFE}}} \times \mathbb{Z}_p^{n_{\mathsf{PHFE}}}$, and every function $f \in \mathcal{F}_{\mathsf{PHFE},d,p,n_{\mathsf{PHFE}}}$ such that $f(\mathsf{PI}, \mathsf{SI}) \in \{0, 1\}$ we have:*

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

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

Definition 7.7 (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 PI along with the outputs $f_j(\mathsf{PI}, \mathsf{SI})$ for every queried function f_j. This is accomplished by requiring that an encryption of $(\mathsf{PI}, \mathsf{SI})$ and keys for functions $\{f_j\}_{j \in Q_{\mathsf{SK}}}$ can be simulated knowing only PI along with $f_j(\mathsf{PI}, \mathsf{SI})$ for all $j \in Q_{\mathsf{SK}}$.

Definition 7.8 (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* $(\mathsf{PI}, \mathsf{SI}) \in \mathbb{Z}_p^{n_{\mathsf{PHFE}}(\lambda)} \times \mathbb{Z}_p^{n_{\mathsf{PHFE}}(\lambda)}$ *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.*

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

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

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* $O(\exp(-\lambda^{\Omega(1)}))$.

7.3 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 sub-linear 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 5:

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

- Parse $\mathsf{PK} = (\mathsf{PHFE.PK}, \mathsf{crs}, p)$.
- Preprocess $\boldsymbol{x}$ using the PRE scheme, $\mathsf{PRE.PreProc}(1^\lambda, 1^{n_{\mathsf{PRE}}}, 1^{m_{\mathsf{PRE}}}, 1^{k_{\mathsf{PRE}}}, p, \boldsymbol{x}) \to (\mathsf{PI}, \mathsf{SI})$.
- 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 $\boldsymbol{y} = (y_1, \ldots, y_T)$ and output $\mathsf{PRE.Dec}(\boldsymbol{y})$.

Figure 5: Description of the FE scheme

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 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 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.

Correctness. Correctness follows from the correctness of PHFE and PRE scheme. The encryption algorithm encrypting $\boldsymbol{x}$ produces $\mathsf{PHFE.Enc}(\mathsf{PHFE.PK}, (\mathsf{PI}, \mathsf{SI}))$ where $(\mathsf{PI}, \mathsf{SI})$ is a preprocessing of input $\boldsymbol{x}$ 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 $\boldsymbol{y} = \mathsf{Encode}(C, (\mathsf{PI}, \mathsf{SI}))$. Finally, the decryption outputs $\mathsf{PRE.Decode}(\boldsymbol{y})$, which is equal to $C(\boldsymbol{x})$ if the PRE scheme is correct.

Sublinearity. We now bound the size of the circuit computing the encryption of an input $\boldsymbol{x} \in \{0,1\}^{n_{\mathsf{FE}}}$. 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 $\boldsymbol{y}$ along with the public input PI. Finally, the security of the PRE scheme ensures that $(\mathsf{PI}, \ \boldsymbol{y})$ is indistinguishable in the case when $\boldsymbol{x}_0$ is encrypted versus the case when $\boldsymbol{x}_1$ is encrypted where $C(\boldsymbol{x}_0) = C(\boldsymbol{x}_1)$. To prove this formally, we list three hybrids, where the first hybrid is an encryption of $\boldsymbol{x}_b$ 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 $\boldsymbol{x}_0, \boldsymbol{x}_1 \in \{0,1\}^{n_{\mathsf{FE}}}$ be the two challenge messages such that $C(\boldsymbol{x}_0) = C(\boldsymbol{x}_1)$. Generate $(\mathsf{PI}, \mathsf{SI})$ using $\mathsf{PRE.PreProc}$ algorithm, while preprocessing $\boldsymbol{x}_b$ 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 $\boldsymbol{y} = \mathsf{PRE.Encode}(C, (\mathsf{PI}, \mathsf{SI}))$.

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 $\{\mathsf{PI}, \{f_i, y_i = f_i(\mathsf{PI}, \mathsf{SI})\}_{i\in[T]}\}$.

$\mathbf{Hybrid}_2$: In this hybrid, generate $(\mathsf{PI}, \boldsymbol{y})$ by first computing $(\mathsf{PI}, \mathsf{SI})$ to preprocess $\boldsymbol{x}_0$, and then computing $\boldsymbol{y} = \mathsf{PRE.Encode}(C, (\mathsf{PI}, \mathsf{SI}))$. 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 $(\mathsf{PI}, \boldsymbol{y})$ are generated. In $\mathbf{Hybrid}_1$, they are generated by using $\boldsymbol{x}_b$, where as in $\mathbf{Hybrid}_2$ they are generated using $\boldsymbol{x}_0$. Note that $C(\boldsymbol{x}_b) = C(\boldsymbol{x}_0)$, and thus the indistinguishability follows from the security of the PRE scheme.

This proves the following result:

Lemma 7.1. *Assuming the existence of a* PHFE *scheme as in Definition 7.5 and a* PRE *scheme as in Definition 6.1, 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 7.3 and the lemma above we have the following Lemma:

Lemma 7.2. *Assuming the existence of a* PHFE *scheme as in Definition 7.5 and a* PRE *scheme as in Definition 6.1, 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 [JLMS19, Wee20, GJLS21], it is shown that PHFE can be constructed using DLIN assumption over prime order symmetric bilinear groups. In Theorem 6.2, it is shown that a PRE scheme can be constructed assuming PRG in NC^0 (Definition 3.3) and δ-LPN assumption for any constant $\delta > 0$ (Definition 3.2). As a consequence, we have the following Theorem:

Theorem 7.4. *If there exists constants* $\delta, \tau > 0$ *such that:*

- δ-LPN *assumption holds (Definition 3.2),*
- *There exists a* PRG *in* NC^0 *with a stretch of* $n^{1+\tau}$ *where* n *is length of the input (Definition 3.3),*
- *The* DLIN *assumption over prime order symmetric bilinear groups 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.*

8 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.

[AB15] Benny Applebaum and Zvika Brakerski. Obfuscating circuits via composite-order graded encoding. In *TCC*, pages 528–556, 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.

[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.

[AJS15a] Prabhanjan Ananth, Abhishek Jain, and Amit Sahai. Achieving compactness generically: Indistinguishability obfuscation from non-compact functional encryption. *IACR Cryptol. ePrint Arch.*, 2015:730, 2015.

[AJS15b] Prabhanjan Ananth, Abhishek Jain, and Amit Sahai. Indistinguishability obfuscation from functional encryption for simple functions. *Eprint*, 730:2015, 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.

[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.

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

[AS17] Prabhanjan Ananth and Amit Sahai. Projective arithmetic functional encryption and indistinguishability obfuscation from degree-5 multilinear maps. In *EUROCRYPT*, 2017.

[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. Cryptology ePrint Archive, Report 2005/417, 2005. `http://eprint.iacr.org/2005/417`.

[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.

[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.

[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.

[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.

[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.

[GGH$^+$13] 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.

[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, *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 555–564. ACM, 2013.

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

[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.

[GVW12] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption with bounded collusions via multi-party computation. In *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, pages 162–179, 2012.

[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.

[IPS09] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Secure arithmetic computation with no honest majority. In *TCC Conference, TCC 2009, San Francisco, CA, USA, March 15-17, 2009. Proceedings*, pages 294–314, 2009.

[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.

[JLS19] Aayush Jain, Huijia Lin, and Amit Sahai. Simplifying constructions and assumptions for $i\mathcal{O}$. *IACR Cryptol. ePrint Arch.*, 2019:1252, 2019.

[JLS21] 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.

[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.

[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.

[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 *CRYPTO*, pages 599–629. Springer, 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. Output-compressing randomized encodings and applications. In *Theory of Cryptography Conference*, pages 96–124. Springer, 2016.

[LT17] Huijia Lin and Stefano Tessaro. Indistinguishability obfuscation from bilinear maps and block-wise local prgs. Cryptology ePrint Archive, Report 2017/250, 2017. `http://eprint.iacr.org/2017/250`.

[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.

[PF79] Nicholas Pippenger and Michael J. Fischer. Relations among complexity measures. *J. ACM*, 26(2):361–381, 1979.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005.

[SS10] Amit Sahai and Hakan Seyalioglu. Worry-free encryption: functional encryption with public keys. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 463–472. ACM, 2010.

[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 Rafael Pass and Krzysztof Pietrzak, editors, *Theory of Cryptography - 18th International Conference, TCC 2020, Durham, NC, USA, November 16-19, 2020, Proceedings, Part I*, volume 12550 of *Lecture Notes in Computer Science*, pages 210–228, 2020.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 1986.