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

Aayush Jain[1], Huijia Lin[2], and Amit Sahai[3]

[1] NTT Research and Carnegie Mellon University
aayushja@andrew.cmu.edu
[2] University of Washington, Seattle
rachel@cs.washington.edu
[3] UCLA
sahai@cs.ucla.edu

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.

1 Introduction

Indistinguishability obfuscation ($i\mathcal{O}$) for general programs computable in polynomial time [12] 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 [19] (see the survey in [20] and references therein), we recently saw the first construction of $i\mathcal{O}$ [30] based on four well-studied assumptions: Learning With Errors (LWE) [38], Decisional Linear assumption (DLIN) [11] over bilinear groups, Learning Parity with Noise (LPN) over $\mathbb{F}_p$ [25], and the existence of a Pseudo-Random Generator (PRG) in NC^0 [21].

While the work of Jain, Lin and Sahai [30] 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 expectations. 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 [30] 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 (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 [2, 10].) Thus, this work removes one, namely LWE, of the two public-key assumptions in [30], making a first step towards understanding the minimal set of assumptions underlying $i\mathcal{O}$.

Lattices vs. (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 [17] 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 [18]. Our work hence yields the first alternative pathways towards these remarkable primitives.

Corollary 2 (Informal). Assume the same assumptions as in the Theorem 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 groups, 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 [30] 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 [7, 32, 37], 3) amplify security via [20], 4) turn secret-ley FE to public-key FE via [13], 5) transform FE with sublinear-size ciphertext to FE with sublinear-time encryption [22, 35], and finally 6) construct $i\mathcal{O}$ from public-key FE for circuits with sublinear-time encryption [6, 14]. 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 [40], we construct a public-key FE for NC^1 with sublinear-time encryption, and then lift the function class from NC^1 to P using Yao's garbled circuits as done in [7, 32, 37], which can be transformed into a public key which implies $i\mathcal{O}$ via [6, 14].

To enable our results, we propose and achieve the new notion of *Preprocessed Randomized Encodings (*PRE*)*. Roughly speaking, PRE allows for preprocessing the input $\mathbf{x}$ and random coins $\mathbf{r}$ into *preprocessed input* $(\mathsf{PI}, \mathsf{SI})$ where PI is public and SI is private, so that, later a randomized encoding of $(f, \mathbf{x})$ 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 $\mathbf{x}$.

We now proceed to an overview of our techniques.

2 Technical Overview

FE Bootstrapping. A common thread in many recent $i\mathcal{O}$ constructions [7, 8, 32, 33, 36] [4, 5, 20, 27, 28, 30, 34] 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 [6, 14].

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 $\mathsf{CT}(\mathbf{x})$ using this secret key reveals only the output $f(\mathbf{x})$, and nothing else about the encrypted input $\mathbf{x}$. To imply $i\mathcal{O}$, it suffices to have FE for NC^1 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 formula of fixed size $\mathsf{poly}(\lambda)$ in the security parameter. 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 [23,39]. An intermediate level of efficiency known as **sublinear size-succinctness** only requires the ciphertext size to be sublinear $|\mathsf{CT}(\mathbf{x})| = \mathsf{poly}(\lambda) \cdot m^{1-\epsilon}$, 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 [22,35]. 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 is generated. So far, from standard assumptions over bilinear maps of order p (e.g. DLIN), prior works [20,28,40] 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-$(O(1), 2)$.

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

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 $C(\mathbf{x})$, 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 $\mathsf{Encode}_C(\mathbf{x}; \mathbf{r})$ have at least locality 4 and hence degree 4 (over $\mathbb{Z}_p$) in $\mathbf{x}$ and $\mathbf{r}$, 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

- $\mathsf{PRE.PreProc}(p, \mathbf{x}) \to (\mathsf{PI}, \mathsf{SI})$. The preprocessing algorithm converts an input $\mathbf{x} \in \{0,1\}^n$ and random tape $\mathbf{r}$ 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, $T_{\mathsf{PreProc}} = m^{1-\epsilon}\mathsf{poly}(\lambda)$.
- $\mathsf{PRE.Encode}(C, \mathsf{PI}, \mathsf{SI}) = \pi$. 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 π. $\mathsf{PRE.Encode}_C$ can be computed by a polynomial mapping over $\mathbb{Z}_p$ with constant degree in PI and degree 2 in SI.
- $\mathsf{PRE.Decode}(\pi) = \mathbf{y}$. The decoding algorithm decodes the output $\mathbf{y} = C(\mathbf{x})$ from π.

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

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

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

The simulation security of the underlying PHFE guarantees that the only information revealed is PI, π. Hence by the indistinguishability security of pRE, FE ciphertexts of inputs $\mathbf{x}_0, \mathbf{x}_1$ 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

We now explain how to construct PRE for NC^1. The main challenges are 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 for NC^1 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 [41], with a PRG in NC^0. The encoding algorithm works as $\pi = \mathsf{Encode}'_C(\mathbf{x}; \mathbf{r}' = \mathsf{PRG}(\mathbf{r}))$, where the random tape has sublinear length $|\mathbf{r}| = m^{1-\tau}\mathsf{poly}(\lambda)$ if the Encode' algorithm uses a linear number of random coins $|\mathbf{r}'| = O(m)\mathsf{poly}(\lambda)$ 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 $(\mathbf{x}, \mathbf{r})$, 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 [30] 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 $\mathbf{x}' = \mathbf{x}, \mathbf{r}$, we will set PI as an encryption $\mathsf{HEEnc}(\mathbf{x}')$ of $\mathbf{x}'$ using a special-purpose homomorphic encryption scheme.
– We set SI to contain the secret key of this homomorphic encryption and some other "preprocessed 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 $\mathbb{F}_p$.

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

where the dimension dim is polynomially related with λ, but relatively small as we describe below. We sample $\mathbf{A} \leftarrow \mathbb{Z}_p^{\dim \times |\mathbf{x}'|}$, $\mathbf{s} \leftarrow \mathbb{Z}_p^{1 \times \dim}$, and the errors $\mathbf{e}$ 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:

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

The LHS of the equation tells us that if we include in SI all degree-d monomials of $\mathbf{s}$, namely, $\mathsf{SI} = (1||\mathbf{s})^{\otimes d}$, then the above quantity can be computed by a polynomial that is degree d in $\mathsf{PI} = (\mathbf{A}, \mathbf{b})$ and in fact linear in SI. By choosing dim to be polynomially smaller than $|\mathbf{x}|$, the above SI will still be sufficiently succinct for our purposes. The RHS of the equation tells us that the error $\mathbf{e}$ is sparse, and h depends only on a constant number d variables, and thus with probability $1 - O(\dim^{-\delta})$, we have $h(\mathbf{x}' + \mathbf{e}) = h(\mathbf{x}')$. 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 $\mathsf{Corr} = \mathsf{Encode}'_C(\mathbf{x}' + \mathbf{e}) - \mathsf{Encode}'_C(\mathbf{x}')$ is sparse: only a $O(\dim^{-\delta})$ fraction of elements in Corr are non-zero. Prior work [30] 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.

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

However, the compression mechanism of [30] 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 $\mathsf{Corr} = \mathsf{Encode}'_C(\mathbf{x}' + \mathbf{e}) - \mathsf{Encode}'_C(\mathbf{x}')$ 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 $\mathbf{e}$ used to encrypt $\mathbf{x}'$ 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 [30], they deal with the first issue by invoking the transformation from size-succinct FE to time-succinct FE [22,35] 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 the security amplification technique of [20] which again relies on LWE [4].

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 $\mathbf{x}$ 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 $\mathbf{x}_1, \ldots, \mathbf{x}_k$ each of dimension n, and we are interested in learning $h(\mathbf{x}_1), \ldots, h(\mathbf{x}_k)$ w.r.t. a polynomial mapping $h : \{0,1\}^n \to \{0,1\}^{m'}$ with constant locality. Can we batch-process $\{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$ into a public and a secret input $(\mathsf{PI}, \mathsf{SI})$ in time sublinear in $m' \cdot k$, such that each $h(\mathbf{x}_i)$ 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

[4] Besides [20], there are other works that contain FE security amplification techniques [4,5,26]. However, it has been recently acknowledged that there is a common issue in these techniques due to an incorrect application of the leakage simulation lemma. The work of [20] circumvents the use leakage simulation lemma, but achieves only weaker security amplification, which nevertheless is still sufficient for constructing $i\mathcal{O}$.

computing polynomials h that have a fixed set of monomials $\mathcal{Q} = \{Q_1, \dots, Q_{m'}\}$. We say that $h(\mathbf{x}_1, \cdots, \mathbf{x}_k)$ has monomial pattern $\mathcal{Q}$ if it has form:

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

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

- $\mathsf{PPE.PreProc}(p, \mathcal{Q}, \mathbf{x}_1, \cdots, \mathbf{x}_k) \to (\mathsf{PI}, \mathsf{SI})$. Given a collection of constant degree-d monomials, $\mathcal{Q} = \{Q_1, \dots, Q_{m'}\}$, the preprocessing algorithm converts a batch of k inputs $\{\mathbf{x}_i \in \{0,1\}^n\}_{i \in [k]}$ 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

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

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

$$\mathsf{Encode}'_C(\mathbf{x}, \mathbf{r}) = \left\{ h_l(\mathbf{x}_1, \cdots, \mathbf{x}_k) = \sum_{i,j} \eta_{l,i,j} Q_j(\mathbf{x}_i) \bmod p \right\}_l$$

$$\text{where } \eta_{l,i,j}\text{'s depend on } C, \text{ but } Q_j\text{'s do not.}$$

We construct such an RE for NC^1, denoted as ARE, using Yao's garbling scheme [41] and a PRG in NC^0. Recall that we consider circuits $C : \{0,1\}^n \to \{0,1\}^{m=m'k}$ where every output bit is computable by a formula of fixed size, say λ. We can divide C into k chunks $C_1, \dots, C_k$ 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.

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

The idea is viewing $\{\mathbf{x}_i = (\mathbf{x}, \mathbf{r}_i)\}$ 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,

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

where U is a universal circuit that takes as input $C_i, \mathbf{x}$ and outputs $C_i(\mathbf{x})$. Now the computation of the garbled tables no longer depend on C_i, neither does the input garbling of $\mathbf{x}$. 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.

Note that our ARE only handles NC^1 circuits because they can be written as formulas. In a formula, every wire w feeds into a single gate g as an input wire. Hence, it suffices to use a PRG with linear stretch to expand the label for wire w into pseudorandom masks used for creating the garbled table for g. If the fan-out were unbounded, we would need a PRF in order to generate the pseudorandom masks for all the gates that wire w feeds into. However, we do not have PRF with constant locality. 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 $|\mathbf{x}' = (\mathbf{x}, \mathbf{r})|$ 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 $\mathcal{Q} = \{Q_i(\mathbf{x}_j)\}_{i \in [m'], j \in [k]}$, 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 $\mathbf{x}_j$, that is,

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

where $\mathbf{A}_j \leftarrow \mathbb{Z}_p^{n \times k}$, $\mathbf{s} \leftarrow \mathbb{Z}_p^{1 \times k}$, and $\mathbf{e}_j \in \mathbb{Z}_p^k$ 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 $\{\mathsf{HEEnc}(Q_i(\mathbf{x}_j))\}_{i,j}$.

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 $\mathbf{s}$, $\mathsf{SI}_0 = (1||\mathbf{s})^{\otimes d}$, so that one can obtain the erroneous outputs $\{Q_i(\mathbf{x}_j + \mathbf{e}_j)\}_{i,j}$. Next, think of the errors Corr as arranged in a $m' \times k$ matrix, where $\mathsf{Corr}[i, j] = Q_i(\mathbf{x}_j + \mathbf{e}_j) - Q_i(\mathbf{x}_j)$. 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 $\{\mathbf{x}_j\}_j$,

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

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 $\{Q_i(\mathbf{x}_j)\}_{i,j}$.

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 $\mathbf{e}_j, \mathbf{x}_j$, 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 ϕ this assignement, $\phi(j) = (j_1, j_2, j_3)$. 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 $\mathbf{M}_{i,\gamma} = \mathbf{U}_{i,\gamma} \cdot \mathbf{V}_{i,\gamma}^{\top}$. 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 $Q_i(\mathbf{x}_j)$ can be computed in degree $(O(1), 2)$ from PI, SI_0 and SI_i as follows:

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

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
$$\mathsf{NZCorr}_i = \{(j,\ \phi(j) = (j_1, j_2, j_3),\ \mathsf{Corr}[i,j]) \mid j \in [k],\ \mathsf{Corr}_i[j] \neq 0\} \ .$$
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 $(j, \phi(j) = (j_1, j_2, j_3), \mathsf{Corr}[i,j])$ 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 $\{\mathbf{e}_j\}_{j\in[k]}$ used to encrypt the inputs $\{\mathbf{x}_j\}_{j\in[k]}$. 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 $\mathbf{x}_j$, such that, the l'th element $x_{j,l}$ is blinded by a non-zero error $e_{j,l} \neq 0$, as well as the values $x_{j,l}, e_{j,l}$ of the input and error elements. Tuples in this list are sorted in ascending order with respect to coordinate j. Note that these lists can be computed in time $O(nk)$.

Now, think of a database that contains all $\{\mathsf{NZInp}_l\}_l$ and inputs $\{\mathbf{x}_j\}_j$, which can be randomly accessed. The second sub-step makes a pass over all monomials $Q_1, \ldots Q_{m'}$. 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 the full version for details, and only highlight some of the tools used in the conversion. We make extensive use of sorting circuits of almost linear size [1] 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 $P_{q,N}^{\mathsf{lookup}}$ indexed with a number $N \in \mathbb{N}$ and a number $q \in \mathbb{N}$ is a program with the following structure: It takes as input q indices $\{i_1, \ldots, i_q\}$ and a database $\mathsf{DB} \in \{0,1\}^N$ and it outputs $\{\mathsf{DB}[i_1], \ldots, \mathsf{DB}[i_q]\}$. We show that this can be implemented efficiently by a circuit:

Lemma 3. *Let $q, N \in \mathbb{N}$. A RAM lookup program $P_{q,N}^{\mathsf{RAM}}$ (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 the full version [29] for how we use the above lemma and other technical details.

Outline This completes are technical overview. In the main body, we present three abstractions PPE, ARE and PRE. In the full version, we show how to combine these abstactions along with a partially hiding FE scheme to build a sublinear functional encryption. 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 $\mathbf{b}$ and $\mathbf{u}$. Matrices will be denoted by capitalized bold-faced letters for such as $\mathbf{A}$ and $\mathbf{M}$. For any $k \in \mathbb{N}$, we denote by the notation $\mathbf{v}^{\otimes k} = \underbrace{\mathbf{v} \otimes \cdots \otimes \mathbf{v}}_{k}$ the standard tensor product. This contains all the monomials in the variables inside $\mathbf{v}$ of degree exactly k.

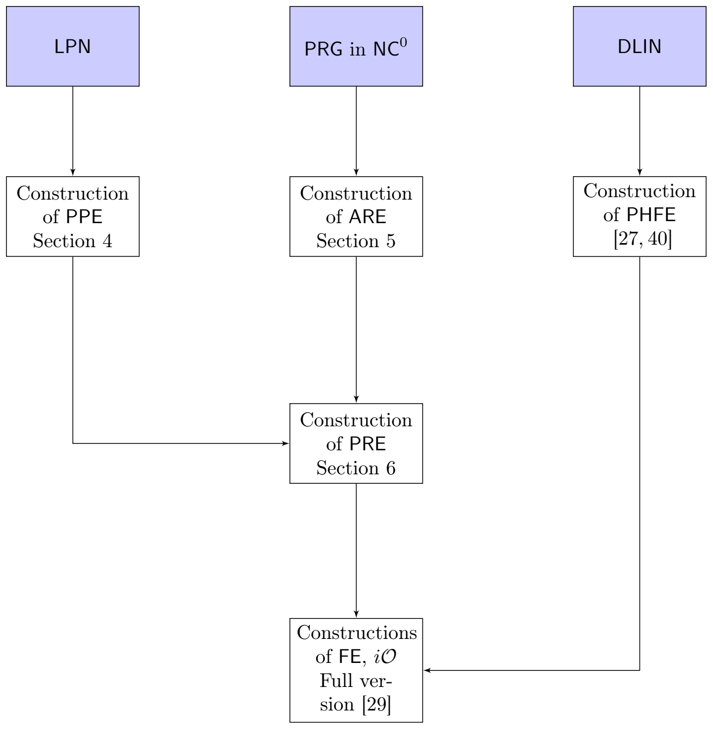
LPN
PRG in NC^0
DLIN
Construction of PPE Section 4
Construction of ARE Section 5
Construction of PHFE [27, 40]
Construction of PRE Section 6
Constructions of FE, $i\mathcal{O}$ Full version [29]

Fig. 1. Flowchart depicting the technical outline.

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

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

Definition 4 (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 [30] for a formal definition of DLIN.

Definition 5 (δ-LPN Assumption, [9, 15, 16, 24]). *Let $\delta \in (0,1)$. We say that the δ-LPN Assumption is true if the following holds: For any constant $\eta_p > 0$, any function $p : \mathbb{N} \to \mathbb{N}$ s.t., for every $\ell \in \mathbb{N}$, $p(\ell)$ is a prime of ℓ^{η_p} bits, any constant $\eta_n > 0$, we set $p = p(\ell)$, $n = n(\ell) = \ell^{\eta_n}$, and $r = r(\ell) = \ell^{-\delta}$, and we require that the following two distributions are computationally indistinguishable:*

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

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

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

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

Definition 6. *(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 7. 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 [3, 31].

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

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

Definition 9 (Polynomial Class $\mathcal{F}_{\mathsf{PPE}}$). *For a constant $d \in \mathbb{N}$, the family of classes of polynomials $\mathcal{F}_{\mathsf{PPE},d} = \{\mathcal{F}_{\mathsf{PPE},d,n_{\mathsf{PPE}},\mathcal{Q},k_{\mathsf{PPE}}}\}_{d \leq n_{\mathsf{PPE}} \in \mathbb{N}, \mathcal{Q} \in \Gamma_{d,n_{\mathsf{PPE}}}, k_{\mathsf{PPE}} \in [\mathbb{N}]}$ consists of polynomials $f \in \mathcal{F}_{\mathsf{PPE},d,n_{\mathsf{PPE}},\mathcal{Q},k_{\mathsf{PPE}}}$ of the following kind: f is defined by a sequence of integers $(\zeta_i^{(j)})_{j \in [k_{\mathsf{PPE}}], i \in [m_{\mathsf{PPE}}]}$. It takes as input $\mathbf{x}$ consisting of k_{PPE} blocks $\mathbf{x} = (\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(k_{\mathsf{PPE}})})$ each of n_{PPE} variables, and has form:*

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

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

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

Definition 10 (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:*

– $(\mathsf{PI}, \mathsf{SI}) \leftarrow \mathsf{PreProc}(1^{n_{\mathsf{PPE}}},\ 1^{k_{\mathsf{PPE}}},\ p,\ \mathcal{Q}, \mathbf{x} \in \mathbb{Z}_p^{n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}})$: *The randomized Preprocessing algorithm takes as input the block length parameter n_{PPE}, the number of blocks parameter k_{PPE}, a prime p, a d-monomial pattern on n_{PPE} variables $\mathcal{Q}$ of size m_{PPE}, and an input $\mathbf{x} \in \mathbb{Z}_p^{n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}}$. It processes it to output two strings, a public string 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 $f(\mathbf{x})$ with high probability.

Definition 11 ((Statistical) Correctness of PPE). *Let $d > 0$ be a constant integer, a PPE scheme for the function class $\mathcal{F}_{d,\mathsf{PPE}}$ satisfies correctness if: For every $k_{\mathsf{PPE}} \in \mathbb{N}$, $n_{\mathsf{PPE}} = k^{\Theta(1)}$, and $\mathcal{Q} \in \Gamma_{d,n_{\mathsf{PPE}}}$ with $m_{\mathsf{PPE}} \geq 1$ sets, any function $f \in \mathcal{F}_{d,\mathsf{PPE},n_{\mathsf{PPE}},\mathcal{Q},k_{\mathsf{PPE}}}$, any prime p and any input $\mathbf{x} \in \mathbb{Z}_p^{n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}}$:*

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

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

Definition 12 (Security of PPE). *Let $d > 0$ be an integer constant. A PPE scheme is secure, if the following holds: Let $\beta > 0$ be any constant and $p : \mathbb{N} \to \mathbb{N}$ be any function that on input an integer r, outputs an r^{β} bit prime. Let $n_{\mathsf{PPE}} = {k_{\mathsf{PPE}}}^{\Theta(1)}$ be any polynomial in k_{PPE}. Let $p = p(k_{\mathsf{PPE}})$ and $\{\mathbf{x}_{k_{\mathsf{PPE}}}\}_{k_{\mathsf{PPE}} \in \mathbb{N}}$ be any ensemble of inputs where each $\mathbf{x}_{k_{\mathsf{PPE}}} \in \mathbb{Z}_p^{n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}}$ and $\{\mathcal{Q}_{k_{\mathsf{PPE}}}\}_{k_{\mathsf{PPE}} \in \mathbb{N}}$ be ensemble of monomial patterns with $\mathcal{Q}_{k_{\mathsf{PPE}}} \in \Gamma_{d,n_{\mathsf{PPE}}}$ with size $m_{\mathsf{PPE}} \geq 1$. Then for $k_{\mathsf{PPE}} \in \mathbb{N}$, it holds that for any probabilistic polynomial time adversary, following* *distributions are computationally indistinguishable with the advantage bounded by* $\mathsf{negl}(k_{\mathsf{PPE}})$.

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

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

Definition 13 (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 14 (Complexity of Evaluation). *Let $d \in \mathbb{N}$ be any constant. We require that* PPE *scheme for $\mathcal{F}_{\mathsf{PPE},d}$ satisfies the following. We require that for every $k_{\mathsf{PPE}} \in \mathbb{N}$, $n_{\mathsf{PPE}} = {k_{\mathsf{PPE}}}^{\Theta(1)}$, and $\Gamma \in \Gamma_{d,n_{\mathsf{PPE}}}$ of size $m_{\mathsf{PPE}} \geq 1$, any prime p, any input $\mathbf{x} \in \mathbb{Z}_p^{n_{\mathsf{PPE}}\cdot k_{\mathsf{PPE}}}$, any pre-processed input* $(\mathsf{PI},\mathsf{SI}) \leftarrow \mathsf{PreProc}(1^{n_{\mathsf{PPE}}},\ 1^{k_{\mathsf{PPE}}},\ p,\ \Gamma,\ \mathbf{x})$, *and any $f \in \mathcal{F}_{d,n_{\mathsf{PPE}},\mathcal{Q},k_{\mathsf{PPE}}}$, the following relation is satisfied:*

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

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 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

$(\mathsf{PI}, \mathsf{SI}) \leftarrow \mathsf{PreProc}(1^{n_{\mathsf{PPE}}}, 1^{k_{\mathsf{PPE}}}, p, \mathcal{Q} = (Q_1, \ldots, Q_{m_{\mathsf{PPE}}}), \mathbf{x})$**:** 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 $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_{k_{\mathsf{PPE}}})$ where each $\mathbf{x}_j \in \mathbb{Z}_p^{n_{\mathsf{PPE}}}$. Parse $\mathbf{x}_j = (x_{j,1}, \ldots, x_{j,n_{\mathsf{PPE}}})$.
- The overall outline is the following: We first show how to sample components $\mathsf{PI}' = (\mathsf{PI}_1, \ldots, \mathsf{PI}_{k_{\mathsf{PPE}}})$, 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 $\mathbf{s} \leftarrow \mathbb{Z}_p^{k_{\mathsf{PPE}}}$. For every $i \in [n_{\mathsf{PPE}}]$, and $j \in [k_{\mathsf{PPE}}]$:
  1. Sample $\mathbf{a}_{j,i} \leftarrow \mathbb{Z}_p^{k_{\mathsf{PPE}}}$.
  2. Sample $e_{j,i} \leftarrow \mathsf{Ber}({k_{\mathsf{PPE}}}^{-\delta}) \cdot \mathbb{Z}_p$. Denote $\mathbf{e}_j = (e_{j,1}, \ldots, e_{j,n_{\mathsf{PPE}}})$.
  3. Compute $b_{j,i} = \langle \mathbf{a}_{j,i}, \mathbf{s} \rangle + e_{j,i} + x_{j,i} \mod p$.

  For $j \in [k_{\mathsf{PPE}}]$, set $\mathsf{PI}_j = \{\mathbf{a}_{j,i}, b_{j,i}\}_{i \in [n_{\mathsf{PPE}}]}$.
- **Sampling** SI**:** SI has $m_{\mathsf{PPE}} + 1$ components. That is, $\mathsf{SI} = (\mathsf{SI}_0, \ldots, \mathsf{SI}_{m_{\mathsf{PPE}}})$. Set $\mathsf{SI}_0 = (1, \mathbf{s})^{\otimes \lceil \frac{d}{2} \rceil}$. We now show how to compute SI_r for $r \in [m_{\mathsf{PPE}}]$.
  1. For $j \in [k_{\mathsf{PPE}}]$, compute $\mathsf{Corr}_{r,j} = \mathsf{Mon}_{Q_r}(\mathbf{x}_j) - \mathsf{Mon}_{Q_r}(\mathbf{x}_j + \mathbf{e}_j)$.
  2. Initialize for every $\gamma \in [t_1]$, matrices $\mathbf{M}_{r,\gamma}$ in $\mathbb{Z}_p^{T \times T}$ with zero entries.
  3. For $j \in [k_{\mathsf{PPE}}]$, compute $\phi(j) = (j_1, (j_2, j_3))$ and set $\mathbf{M}_{r,j_1}[j_2, j_3] = \mathsf{Corr}_{r,j}$. If any matrix $\mathbf{M}_{r,\gamma}$ for $\gamma \in [t_1]$, has more than t non-zero entries, then set $\mathsf{flag}_r = 0$. Otherwise, set $\mathsf{flag}_r = 1$.
  4. If $\mathsf{flag}_r = 1$, then, for $\gamma \in [t_1]$, compute matrices $\mathbf{U}_{r,\gamma}, \mathbf{V}_{r,\gamma}^{\top} \in \mathbb{Z}_p^{T \times t}$ such that $\mathbf{M}_{r,\gamma} = \mathbf{U}_{r,\gamma} \cdot \mathbf{V}_{r,\gamma}$. Otherwise for every $\gamma \in [t_1]$, set $\mathbf{U}_{r,\gamma}, \mathbf{V}_{r,\gamma}$ to be matrices with zero-entries.
  5. Set $\mathsf{SI}_r = \{\mathbf{U}_{r,\gamma}, \mathbf{V}_{r,\gamma}\}_{\gamma \in [t_1]}$.
- **Sampling** flag**:** For every $i \in [n_{\mathsf{PPE}}]$, let $\mathsf{Set}_i = \{j \in [k_{\mathsf{PPE}}] | e_{j,i} \neq 0\}$. If any of these sets have size outside the range $[{k_{\mathsf{PPE}}}^{1-\delta} - t{k_{\mathsf{PPE}}}^{\frac{1-\delta}{2}}, {k_{\mathsf{PPE}}}^{1-\delta} + t{k_{\mathsf{PPE}}}^{\frac{1-\delta}{2}}]$, 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 $\mathsf{PI}_j = \{\mathbf{a}_{j,i}, b_{j,i}\}_{i \in [n_{\mathsf{PPE}}]}$. Similarly, parse $\mathsf{SI} = (\mathsf{SI}_0, \ldots, \mathsf{SI}_{m_{\mathsf{PPE}}})$. Here $\mathsf{SI}_0 = (1, \mathbf{s})^{\otimes \lceil \frac{d}{2} \rceil}$ and $\mathsf{SI}_r = \{\mathbf{U}_{r,\gamma}, \mathbf{V}_{r,\gamma}\}_{\gamma \in [t_1]}$ for $r \in [m_{\mathsf{PPE}}]$. Parse $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_{k_{\mathsf{PPE}}})$ and $f(\mathbf{x}) = \sum_{r \in [m_{\mathsf{PPE}}], j \in [k_{\mathsf{PPE}}]} \mu_{r,j} \mathsf{Mon}_{Q_r}(\mathbf{x}_j)$ for $\mu_{r,j} \in \mathbb{Z}$. Output:

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

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

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

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

Remark 15. 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.

In the full version [29], we argue correctness, efficiency, complexity and security properties.

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

Theorem 16. *Assuming δ-LPN assumption (Definition 5) holds for any constant $\delta > 0$, then there exists a PPE scheme satisfying Definition 10. 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 Boolean formula of size λ (circuits where each gate has a single fan-out). Such an ARE scheme has the following syntax:

Definition 17 (Syntax of ARE). *An ARE scheme consists of the following p.p.t. algorithms:*

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

An ARE scheme satisfies the following properties.

Definition 18 ((Perfect) Correctness of ARE). *A ARE scheme for the function class $\mathcal{F}_{\mathsf{ARE}}$ satisfies correctness if: For every polynomials $n_{\mathsf{ARE}}(\cdot), m_{\mathsf{ARE}}(\cdot), k_{\mathsf{ARE}}(\cdot)$, every $\lambda \in \mathbb{N}$, let $n_{\mathsf{ARE}} = n_{\mathsf{ARE}}(\lambda), m_{\mathsf{ARE}} = m_{\mathsf{ARE}}(\lambda), k_{\mathsf{ARE}} = k_{\mathsf{ARE}}(\lambda)$. Then, for every $\mathbf{x} \in \{0,1\}^{n_{\mathsf{ARE}}}$, $C \in \mathcal{F}_{\mathsf{ARE},n_{\mathsf{ARE}},m_{\mathsf{ARE}},k_{\mathsf{ARE}},\lambda}$:*

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

Definition 19 (Indistinguishability Security). *We say that ARE scheme is secure if the following holds: Let $\lambda \in \mathbb{N}$ be the security parameter, and $n_{\mathsf{ARE}}, m_{\mathsf{ARE}}, k_{\mathsf{ARE}} = \Theta(\lambda^{\Theta(1)})$ be polynomials in λ. For every sequence $\{C, \mathbf{x}_0, \mathbf{x}_1\}_\lambda$ where $\mathbf{x}_0, \mathbf{x}_1 \in \{0,1\}^{n_{\mathsf{ARE}}}$ and $C \in \mathcal{F}_{\mathsf{ARE},n_{\mathsf{ARE}},m_{\mathsf{ARE}},k_{\mathsf{ARE}},\lambda}$ with $C(\mathbf{x}_0) = C(\mathbf{x}_1)$, it holds that for $\lambda \in \mathbb{N}$ the following distributions are computationally indistinguishable*

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

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

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

Definition 20 (Efficiency). *We require that there exists constants $d \in \mathbb{N}, c_1, c_2 > 0$, such that the following holds. For any $\lambda \in \mathbb{N}$ and any $n_{\mathsf{ARE}}, k_{\mathsf{ARE}}, m_{\mathsf{ARE}} = \lambda^{\Omega(1)}$, there exists an efficiently samplable degree d-monomial pattern $\mathcal{Q}$ of size $m'_{\mathsf{ARE}} = O((n_{\mathsf{ARE}} + m_{\mathsf{ARE}})\lambda^{c_1})$ such that for any circuit $C \in \mathcal{F}_{\mathsf{ARE},n_{\mathsf{ARE}},m_{\mathsf{ARE}},k_{\mathsf{ARE}},\lambda}$ and input $\mathbf{x} \in \{0,1\}^{n_{\mathsf{ARE}}}$, $\mathsf{Encode}(C, \mathbf{x}; \mathbf{r}) \rightarrow \mathbf{y} \in \{0,1\}^T$ satisfies the following requirements:*

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

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

5.1 Construction Details

In Figure 2, we now give the formal construction of the ARE scheme. We establish some useful notations and recall the tools we need.

Notation and Ingredients: $\lambda \in \mathbb{N}$ is the security parameter, $n_{\mathsf{ARE}}, m_{\mathsf{ARE}}, k_{\mathsf{ARE}}$ are parameters associated with the function class $\mathcal{F}_{\mathsf{ARE},n_{\mathsf{ARE}},m_{\mathsf{ARE}},k_{\mathsf{ARE}}}$.

Tool: We use a PRG in NC^0 (denoted by G) that stretches $t^{1-\epsilon}$ bits to t bits (t is set below). We also use a PRG in NC^0 (denoted by H) that stretches λ bits to $2 \cdot \lambda + 2$ bits. Denote by H_0 the function that computes first half of the output of H and by H_1 the function that computes the other half.
We set $n'_{\mathsf{ARE}} = (n_{\mathsf{ARE}} + m^{1-\epsilon}_{\mathsf{ARE}})\mathsf{poly}(\lambda)$ for a large enough polynomial poly. We will set $t = n'_{\mathsf{ARE}} - n_{\mathsf{ARE}}$.
Universal formula implementing a formula: 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 formula for evaluating Boolean formulas with n_{ARE}-bit inputs, m_{ARE}-bit outputs, and size $m_{\mathsf{ARE}} \cdot \lambda$. In particular, $U(C_i, \mathbf{x}) = C_i(\mathbf{x})$ for circuits C_i and input $\mathbf{x}_i$ satisfying the requirements. The size of each U is $\widetilde{O}(n_{\mathsf{ARE}} + m_{\mathsf{ARE}})$, where $\widetilde{O}$ hides polynomial factors in $\log n_{\mathsf{ARE}}, \log n_{\mathsf{ARE}}, \lambda$. Since $n_{\mathsf{ARE}}, m_{\mathsf{ARE}}$ are all polynomials in λ, we ignore its dependence on logarithmic factors.

In full version [29], we discuss why all the properties are satisfied. Thus, we have the following theorem:

Theorem 21. *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 6), then there exists an* ARE *scheme satisfying Definition 17. 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 Boolean formulas 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 Boolean formula of size λ.

Definition 22 (Syntax of Preprocessed Randomized Encoding). *A preprocessed randomized encoding scheme* PRE *for the function class* $\mathcal{F}_{\mathsf{PRE}}$ *contains the following polynomial time algorithms:*

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

The ARE scheme

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

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

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

  Above $C_{\kappa,i}$ is i^{th} bit of the circuit description.
- Compute garbled tables for U. That is, for every gate gate in U with input wires w_1, w_2 and output wire w_3, output the following garbled table.

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

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

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

Fig. 2. ARE Scheme Description

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

Remark 23. Note that we could have defined the primitive without a parameter k_{PRE} by considering formulas 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 24 (Correctness). *We say that* PRE *is correct if the following holds: For every* $\lambda \in \mathbb{N}$, $n_{\mathsf{PRE}}, m_{\mathsf{PRE}}, k_{\mathsf{PRE}} = \Theta(\lambda^{\Theta(1)})$, *$p$ a prime,* $\mathbf{x} \in \{0,1\}^{n_{\mathsf{PRE}}}$, *and* $C \in \mathcal{F}_{\mathsf{PRE},n_{\mathsf{PRE}},m_{\mathsf{PRE}},k_{\mathsf{PRE}},\lambda}$.

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

Definition 25 (Indistinguishability Security). *We say that* PRE *scheme is secure if the following holds: Let* $\beta, c_1, c_2, c_3 > 0$ *be arbitrary constants, and* $p : \mathbb{N} \to \mathbb{N}$ *be any function that takes as input any integer r and outputs a* r^β *bit prime and* $n_{\mathsf{PRE}}(r) = r^{c_1}$, $m_{\mathsf{PRE}}(r) = r^{c_2}$ *and* $k_{\mathsf{PRE}} = r^{c_3}$ *be three polynomials. Let* $\{C, \mathbf{x}_0, \mathbf{x}_1\}_{\lambda \in \mathbb{N}}$ *be any ensemble where* $\mathbf{x}_0, \mathbf{x}_1 \in \{0,1\}^{n_{\mathsf{PRE}}(\lambda)}$ *and* $C \in \mathcal{F}_{\mathsf{PRE},n_{\mathsf{PRE}}(\lambda),m_{\mathsf{PRE}}(\lambda),k_{\mathsf{PRE}}(\lambda),\lambda}$ *with* $\mathbf{y} = C(\mathbf{x}_0) = C(\mathbf{x}_1)$. *Then it holds that for any* $\lambda \in \mathbb{N}$, *and letting* $p = p(\lambda)$, $n_{\mathsf{PRE}} = n_{\mathsf{PRE}}(\lambda)$, $m_{\mathsf{PRE}} = m_{\mathsf{PRE}}(\lambda)$ *and* $k_{\mathsf{PRE}} = k_{\mathsf{PRE}}(\lambda)$ *it holds that the following distributions are computationally indistinguishable*

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

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

The Efficiency and Complexity Requirements

Definition 26 (Sublinear Efficiency of PRE). *We require that there exists a polynomial* poly *and constants* $c_1, c_2, c_3 > 0$ *such that for every polynomials* $n_{\mathsf{PRE}}, m_{\mathsf{PRE}}$ *and* k_{PRE} *and every security parameter* $\lambda \in \mathbb{N}$, *every prime p, the (randomized) circuit* $D(\cdot)$ *that on input* $\mathbf{x} \in \{0,1\}^{n_{\mathsf{PRE}}}$ *computes* $\mathsf{PRE.PreProc}(1^\lambda, 1^{n_{\mathsf{PRE}}}, 1^{m_{\mathsf{PRE}}}, 1^{k_{\mathsf{PRE}}}, p, \mathbf{x})$ *has size bounded by* $((n_{\mathsf{PRE}} + {m_{\mathsf{PRE}}}^{1-c_1}){k_{\mathsf{PRE}}}^{c_2} + m_{\mathsf{PRE}}{k_{\mathsf{PRE}}}^{1-c_3})\mathsf{poly}(\lambda, \log p)$.

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

Definition 27 (Complexity of Encoding). *We require that for every polynomials $n_{\mathsf{PRE}}, m_{\mathsf{PRE}}, k_{\mathsf{PRE}}$, every security parameter $\lambda \in \mathbb{N}$, every $C \in \mathcal{F}_{\mathsf{PRE}, n_{\mathsf{PRE}}, m_{\mathsf{PRE}}, k_{\mathsf{PRE}}, \lambda}$, and every prime p, there exists a polynomial mapping f satisfying the following:*

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

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

– *There is a universal constant $d \in \mathbb{N}$ independent of all parameters, s.t., f has degree d in PI and degree 2 in SI.*
– *f can be uniformly and efficiently generated from $\lambda, n_{\mathsf{PRE}}, m_{\mathsf{PRE}}, k_{\mathsf{PRE}}, p, C$.*

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 $g_f(\cdot) = \mathsf{PPE.Eval}(f, \cdot) \bmod p$ used to evaluate any polynomial $f \in \mathcal{F}_{d, \mathsf{PPE}, n_{\mathsf{PPE}}, \mathcal{Q}, k_{\mathsf{PPE}}}$.

We now describe our construction in Figure 3:

In the full version, we argue various properties associated with a PRE scheme. Thus, we have the following theorem:

The PRE scheme

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

- Sample uniformly randomness $\mathbf{r}_1, \ldots, \mathbf{r}_{k_{\mathsf{ARE}}} \in \{0,1\}^{n'_{\mathsf{ARE}} - n_{\mathsf{ARE}}}$ used for running $\mathsf{ARE.Encode}(\cdot, \mathbf{x}, \mathbf{r})$. Set $\mathbf{a}_i = (\mathbf{x}, \mathbf{r}_i)$ for $i \in [k_{\mathsf{ARE}}]$. Here $\mathbf{a}_i \in \{0,1\}^{n'_{\mathsf{ARE}} = n_{\mathsf{PPE}}}$.
- Compute $(\mathsf{PI}, \mathsf{SI}) \leftarrow \mathsf{PPE.PreProc}(1^{n_{\mathsf{PPE}}}, 1^{k_{\mathsf{PPE}}}, p, \mathcal{Q}, \mathbf{a})$. 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 $\mathbf{y} = (y_1, \ldots, y_T)$.

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

Fig. 3. The Description of the PRE scheme.

Theorem 28. *Assume that there exists two constant $\delta, \epsilon > 0$ such that:*

- *δ-LPN assumption (Definition 5) holds,*
- *There exists a PRG in NC^0 with a stretch $n^{1+\epsilon}$ where n is the length of the input (Definition 6),*

Then, there exists a PRE scheme (Definition 22). Further, assuming the underlying assumptions are subexponentially secure, then so is the resulting PRE scheme.

7 Summing Up

In the full version [29], we use a PRE scheme and combine it with a partially hiding functional encryption, to build a sublinear functional encryption for Boolean formulas and then bootstrap it to $i\mathcal{O}$ using prior results. Thus, we prove:

Theorem 29. *If there exists constants $\delta, \tau > 0$ such that:*

- *δ-LPN assumption holds (Definition 5),*
- *There exists a PRG in NC^0 with a stretch of $n^{1+\tau}$ where n is length of the input (Definition 6),*
- *The DLIN assumption over prime order symmetric bilinear groups holds.*

Then, there exists a sublinear functional encryption scheme for circuits. Further if the underlying assumptions are subexponentially secure, then there exists a secure indistinguishability obfuscation for all circuits.

8 Acknowledgements

Aayush Jain is supported by NTT Research, a grant from CyLab security and privacy institute and a start-up package by the computer science department at CMU.

Huijia Lin is supported by NSF grants CNS- 2026774, CNS-1936825 (CAREER), Simons JP Morgan Faculty Award, and a Simons collaboration grant on algorithmic fairness.

Amit Sahai is supported in part from a Simons Investigator Award, DARPA SIEVE award, NTT Research, NSF Frontier Award 1413955, BSF grant 2012378, a Xerox Faculty Research Award, a Google Faculty Research Award, and an Okawa Foundation Research Grant. This material is based upon work supported by the Defense Advanced Research Projects Agency through Award HR00112020024.

The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense, DARPA, ARO, Simons, Intel, Okawa Foundation, ODNI, IARPA, DIMACS, BSF, Xerox, the National Science Foundation, NTT Research, Google, J.P. Morgan or the U.S. Government.

References

1. Ajtai, M., Komlós, J., Szemerédi, E.: An $O(n \log n)$ sorting network. In: 15th ACM STOC. pp. 1–9. ACM Press (Apr 1983)
2. Alekhnovich, M.: More on average case vs approximation complexity. In: 44th FOCS. pp. 298–307. IEEE Computer Society Press (Oct 2003)
3. Ananth, P., Badrinarayanan, S., Jain, A., Manohar, N., Sahai, A.: From FE combiners to secure MPC and back. In: Hofheinz, D., Rosen, A. (eds.) TCC 2019, Part I. LNCS, vol. 11891, pp. 199–228. Springer, Heidelberg (Dec 2019)
4. Ananth, P., Jain, A., Lin, H., Matt, C., Sahai, A.: Indistinguishability obfuscation without multilinear maps: New paradigms via low degree weak pseudorandomness and security amplification. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part III. LNCS, vol. 11694, pp. 284–332. Springer, Heidelberg (Aug 2019)
5. Ananth, P., Jain, A., Sahai, A.: Indistinguishability obfuscation without multilinear maps: io from lwe, bilinear maps, and weak pseudorandomness. IACR Cryptology ePrint Archive 2018, 615 (2018)
6. Ananth, P., Jain, A.: Indistinguishability obfuscation from compact functional encryption. In: Gennaro, R., Robshaw, M.J.B. (eds.) CRYPTO 2015, Part I. LNCS, vol. 9215, pp. 308–326. Springer, Heidelberg (Aug 2015)
7. Ananth, P., Jain, A., Sahai, A.: Indistinguishability obfuscation from functional encryption for simple functions. Eprint 730, 2015 (2015)
8. Ananth, P., Sahai, A.: Projective arithmetic functional encryption and indistinguishability obfuscation from degree-5 multilinear maps. In: EUROCRYPT (2017)
9. Applebaum, B., Avron, J., Brzuska, C.: Arithmetic cryptography: Extended abstract. In: Roughgarden, T. (ed.) ITCS 2015. pp. 143–151. ACM (Jan 2015)
10. Applebaum, B., Brakerski, Z.: Obfuscating circuits via composite-order graded encoding. In: TCC. pp. 528–556 (2015)

11. Ballard, L., Green, M., de Medeiros, B., Monrose, F.: Correlation-resistant storage via keyword-searchable encryption. Cryptology ePrint Archive, Report 2005/417 (2005), `http://eprint.iacr.org/2005/417`
12. Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S.P., Yang, K.: On the (im)possibility of obfuscating programs. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 1–18. Springer, Heidelberg (Aug 2001)
13. Bitansky, N., Nishimaki, R., Passelègue, A., Wichs, D.: From cryptomania to obfustopia through secret-key functional encryption. In: Hirt, M., Smith, A.D. (eds.) TCC 2016-B, Part II. LNCS, vol. 9986, pp. 391–418. Springer, Heidelberg (Oct / Nov 2016)
14. Bitansky, N., Vaikuntanathan, V.: Indistinguishability obfuscation from functional encryption. In: Guruswami, V. (ed.) 56th FOCS. pp. 171–190. IEEE Computer Society Press (Oct 2015)
15. Blum, A., Furst, M.L., Kearns, M.J., Lipton, R.J.: Cryptographic primitives based on hard learning problems. In: Stinson, D.R. (ed.) CRYPTO'93. LNCS, vol. 773, pp. 278–291. Springer, Heidelberg (Aug 1994)
16. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y.: Compressing vector OLE. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) ACM CCS 2018. pp. 896–912. ACM Press (Oct 2018)
17. Canetti, R., Lin, H., Tessaro, S., Vaikuntanathan, V.: Obfuscation of probabilistic circuits and applications. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015, Part II. LNCS, vol. 9015, pp. 468–497. Springer, Heidelberg (Mar 2015)
18. van Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V.: Fully homomorphic encryption over the integers. In: Gilbert, H. (ed.) 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. Lecture Notes in Computer Science, vol. 6110, pp. 24–43. Springer (2010)
19. Garg, S., Gentry, C., Halevi, S., Raykova, M., Sahai, A., Waters, B.: Candidate indistinguishability obfuscation and functional encryption for all circuits. In: 54th FOCS. pp. 40–49. IEEE Computer Society Press (Oct 2013)
20. Gay, R., Jain, A., Lin, H., Sahai, A.: Indistinguishability obfuscation from simple-to-state hard problems: New assumptions, new techniques, and simplification. In: EUROCRYPT. Lecture Notes in Computer Science, vol. 12698, pp. 97–126. Springer (2021)
21. Goldreich, O.: Candidate one-way functions based on expander graphs. Electronic Colloquium on Computational Complexity (ECCC) 7(90) (2000)
22. Goldwasser, S., Kalai, Y.T., Popa, R.A., Vaikuntanathan, V., Zeldovich, N.: Reusable garbled circuits and succinct functional encryption. In: Boneh, D., Roughgarden, T., Feigenbaum, J. (eds.) Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013. pp. 555–564. ACM (2013), `http://doi.acm.org/10.1145/2488608.2488678`
23. Gorbunov, S., Vaikuntanathan, V., Wee, H.: 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. pp. 162–179 (2012)
24. Ishai, Y., Prabhakaran, M., Sahai, A.: Founding cryptography on oblivious transfer - efficiently. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 572–591. Springer, Heidelberg (Aug 2008)

25. Ishai, Y., Prabhakaran, M., Sahai, A.: Secure arithmetic computation with no honest majority. In: TCC Conference, TCC 2009, San Francisco, CA, USA, March 15-17, 2009. Proceedings. pp. 294–314 (2009)
26. Jain, A., Korb, A., Manohar, N., Sahai, A.: Amplifying functional encryption, unconditionally. CRYPTO 2020 (2020)
27. Jain, A., Lin, H., Matt, C., Sahai, A.: How to leverage hardness of constant-degree expanding polynomials overa $\mathbb{R}$ to build $i\mathcal{O}$. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part I. LNCS, vol. 11476, pp. 251–281. Springer, Heidelberg (May 2019)
28. Jain, A., Lin, H., Sahai, A.: Simplifying constructions and assumptions for $i\mathcal{O}$. IACR Cryptol. ePrint Arch. 2019, 1252 (2019), `https://eprint.iacr.org/2019/1252`
29. Jain, A., Lin, H., Sahai, A.: Indistinguishability obfuscation from LPN over f_p, dlin, and prgs in ncˆ0. IACR Cryptol. ePrint Arch. p. 1334 (2021), `https://eprint.iacr.org/2021/1334`
30. Jain, A., Lin, H., Sahai, A.: Indistinguishability obfuscation from well-founded assumptions. In: Khuller, S., Williams, V.V. (eds.) STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021. pp. 60–73. ACM (2021)
31. Jain, A., Manohar, N., Sahai, A.: Combiners for functional encryption, unconditionally. In: Rijmen, V., Ishai, Y. (eds.) EUROCRYPT 2020, Part I. pp. 141–168. LNCS, Springer, Heidelberg (May 2020)
32. Lin, H.: Indistinguishability obfuscation from constant-degree graded encoding schemes. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part I. LNCS, vol. 9665, pp. 28–57. Springer, Heidelberg (May 2016)
33. Lin, H.: Indistinguishability obfuscation from sxdh on 5-linear maps and locality-5 prgs. In: CRYPTO. pp. 599–629. Springer (2017)
34. Lin, H., Matt, C.: Pseudo flawed-smudging generators and their application to indistinguishability obfuscation. IACR Cryptology ePrint Archive 2018, 646 (2018)
35. Lin, H., Pass, R., Seth, K., Telang, S.: Output-compressing randomized encodings and applications. In: Theory of Cryptography Conference. pp. 96–124. Springer (2016)
36. Lin, H., Tessaro, S.: Indistinguishability obfuscation from bilinear maps and block-wise local prgs. Cryptology ePrint Archive, Report 2017/250 (2017), `http://eprint.iacr.org/2017/250`
37. Lin, H., Vaikuntanathan, V.: Indistinguishability obfuscation from DDH-like assumptions on constant-degree graded encodings. In: Dinur, I. (ed.) 57th FOCS. pp. 11–20. IEEE Computer Society Press (Oct 2016)
38. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Gabow, H.N., Fagin, R. (eds.) 37th ACM STOC. pp. 84–93. ACM Press (May 2005)
39. Sahai, A., Seyalioglu, H.: Worry-free encryption: functional encryption with public keys. In: Proceedings of the 17th ACM conference on Computer and communications security. pp. 463–472. ACM (2010)
40. Wee, H.: Functional encryption for quadratic functions from k-lin, revisited. In: Pass, R., Pietrzak, K. (eds.) TCC 20. Lecture Notes in Computer Science, vol. 12550, pp. 210–228 (2020)
41. Yao, A.C.C.: How to generate and exchange secrets (extended abstract). In: FOCS. pp. 162–167 (1986)