Indistinguishability Obfuscation from Bilinear Maps and LPN Variants

Seyoon Ragavan[*] Neekon Vafa[†] Vinod Vaikuntanathan[‡]

May 31, 2024

Abstract

We construct an indistinguishability obfuscation (IO) scheme from the sub-exponential hardness of the decisional linear problem on bilinear groups together with two variants of the learning parity with noise (LPN) problem, namely large-field LPN and (binary-field) sparse LPN. This removes the need to assume the existence pseudorandom generators (PRGs) in NC^0 with polynomial stretch from the state-of-the-art construction of IO (Jain, Lin, and Sahai, EUROCRYPT 2022). As an intermediate step in our construction, we abstract away a notion of structured-seed polynomial-stretch PRGs in NC^0 which suffices for IO and is implied by both sparse LPN and the existence of polynomial-stretch PRGs in NC^0.

As immediate applications, from the sub-exponential hardness of the decisional linear assumption on bilinear groups, large-field LPN, and sparse LPN, we get alternative constructions of (a) fully homomorphic encryption (FHE) without lattices or circular security assumptions (Canetti, Lin, Tessaro, and Vaikuntanathan, TCC 2015), and (b) perfect zero-knowledge adaptively-sound succinct non-interactive arguments (SNARGs) for NP (Waters and Wu, STOC 2024).

[*]MIT. Email: `sragavan@mit.edu`
[†]MIT. Email: `nvafa@mit.edu`
[‡]MIT. Email: `vinodv@mit.edu`

Contents

1 Introduction

Indistinguishability obfuscation ($i\mathcal{O}$) [BGI+12] is a probabilistic polynomial-time algorithm $\mathcal{O}$ that takes as input a circuit C and outputs an (obfuscated) circuit $\hat{C} \leftarrow \mathcal{O}(C;r)$ satisfying three properties: (a) functionality: C and $\hat{C}$ compute the same function; (b) efficiency: $\mathcal{O}$ runs in polynomial time; in particular, the size of $\mathcal{O}(C)$ is polynomially related to that of C; and (c) security: for any two circuits C_1 and C_2 that compute the same function (and have the same size), the distributions $\mathcal{O}(C_1)$ and $\mathcal{O}(C_2)$ are computationally indistinguishable. While the utility of the $i\mathcal{O}$ definition was not clear for a while, a large body of results building on the breakthrough work of Sahai and Waters [SW14] changed all of that and showed that $i\mathcal{O}$ is indeed a "central hub" of cryptography, implying the existence of a vast swathe of cryptographic primitives, both old and new, as well as new insights in complexity theory.

The first $i\mathcal{O}$ candidate (without a security reduction) was constructed by Garg, Gentry, Halevi, Raykova, Sahai, and Waters in [GGH+13]. Nearly a decade of work later, Jain, Lin, and Sahai [JLS21] showed how to construct IO assuming the sub-exponential hardness of four rather different, but reasonable and well-founded, computational problems:

1. the decisional linear problem on symmetric bilinear groups of prime order p;[1]
2. the learning parity with noise ($\mathsf{FieldLPN}$) problem over exponentially large fields $\mathbb{Z}_p$ where the noise rate is $1/n^{\delta}$, n being the LPN dimension and $\delta > 0$ being any constant;
3. the existence of a Boolean pseudorandom generator in NC^0 with *polynomial stretch*, namely stretching n bits to $n^{1+\epsilon}$ bits for some constant $\epsilon > 0$; and
4. the learning with errors (LWE) problem with a sub-exponential modulus-to-noise ratio.

The subsequent work by the same authors [JLS22] eliminated assumption (4), namely the LWE assumption. Given how central $i\mathcal{O}$ is to theoretical computer science as a whole, it is important to understand the minimal assumptions required to construct it.

This Work: $i\mathcal{O}$ from Bilinear Maps and LPN Variants. We make progress in constructing $i\mathcal{O}$ from weaker assumptions by getting rid of assumption (3) above, namely the existence of a pseudorandom generator in NC^0 with polynomial stretch. Instead, our construction relies on assumption (1), together with two variants of the learning parity with noise problem: the first being assumption (2) as used in [JLS21, JLS22], and the second is the hardness of the sparse learning parity with noise ($\mathsf{SparseLPN}$) problem over $\mathbb{Z}_2$.

> The sub-exponential sparse learning parity with noise ($\mathsf{SparseLPN}$) assumption says that there exist constants $\epsilon, \delta \in \mathbb{R}^+, t \in \mathbb{N}$ and, letting $m = n^{1+\epsilon}$ and $\eta = 1/n^{\delta}$, a distribution[2] $\mathcal{D}$ over matrices $\mathbf{A} \in \mathbb{Z}_2^{m\times n}$ with t-sparse rows such that for any p.p.t. adversary $\mathcal{A}$,
>
> $$\left| \Pr_{\mathbf{A}\leftarrow\mathcal{D},\mathbf{s}\leftarrow\mathbb{Z}_2^n,\mathbf{e}\leftarrow\mathsf{Bern}(\eta)^m}[\mathcal{A}(\mathbf{A},\mathbf{As}+\mathbf{e} \mod 2)=1] - \Pr_{\mathbf{A}\leftarrow\mathcal{D},\mathbf{u}\leftarrow\mathbb{Z}_2^m}[\mathcal{A}(\mathbf{A},\mathbf{u})=1]\right| \le \exp(-n^{\Omega(1)}).$$

[1] [JLS21] relies on the symmetric external Diffie-Hellman (SXDH) assumption on asymmetric bilinear prime-order groups, but [JLS22] as well as our work rely on the DLIN assumption on symmetric bilinear prime-order groups.

[2] For technical reasons, in the sub-exponential regime, we do not require $\mathcal{D}$ to be efficiently sampleable, but we do require that there be an efficient sampler $\mathcal{D}'$ that is in some sense $\Omega(1)$-close to $\mathcal{D}$. In the negligible but not sub-exponential regime, we can (plausibly) sample good $\mathbf{A}$ efficiently by using a sampler of Applebaum and Kachlon [AK23]. For more details, see Sections 2.3 and 3.4 and Appendix A.

A few words about the SparseLPN assumption are in order. First of all, variants of the SparseLPN problem have been studied in many works in the cryptography, constraint satisfaction problem, and average-case complexity literature [Gol00, CM01, Fei02, Ale11, MST06, AOW15, AL18, KMOW17, ADI+17]. In fact, it has been used to build several cryptographic objects, including public-key encryption, cryptography with constant computational overhead, multi-party homomorphic secret sharing, and pseudorandom correlation generators [Ale11, AIK08, IKOS08, ABW10, BCGI18, DIJL23, DJ24]. Secondly, we note that the parameter regime we require for our SparseLPN assumption seems quantitatively weaker than the one required for public-key encryption. In particular, Applebaum, Barak, and Wigderson [ABW10] show how to build public-key encryption from a variant of SparseLPN where the sparsity, noise rate, and number of samples are all related.[3] In comparison, we do not require any such relation between these parameters.

With that said, we now state our main theorem.

Theorem 1 (Informal Version of Corollary 7.11)**.** *Under the sub-exponential hardness of assumptions (1) and (2) and the* SparseLPN *assumption, there exists an $i\mathcal{O}$ scheme.*

Similarly to SparseLPN, assumption (2), namely FieldLPN for any inverse polynomial noise rate, is also weaker than public-key encryption to the best of our knowledge, as the (natural finite-field analog of the) public-key encryption in [Ale11] requires $O(1/\sqrt{n})$ noise rate. Therefore, the only one of these assumptions that implies public-key encryption is assumption (1), the decisional linear problem on symmetric bilinear groups of prime order.[4]

Isn't This Easy? At first sight, it might appear that the SparseLPN assumption directly gives us a polynomial-stretch pseudorandom generator computable in NC^0 (assumption 3) which, together with assumptions 1 and 2, is sufficient for the [JLS22] construction. In fact, Applebaum, Ishai, and Kushilevitz [AIK08] show how to build a *linear-stretch* PRG in NC^0 from this assumption. For [JLS22], we need a polynomial-stretch PRG in NC^0, so it would be natural to try to extend the result of [AIK08] to handle polynomial stretch. Unfortunately, as [AIK08] mention, their techniques do not yield a PRG in NC^0 with superlinear stretch, let alone polynomial stretch.

Let us see what goes wrong with the direct construction. To be more precise, consider the function (family) $g_{\mathbf{A}} : \mathbb{F}_2^n \times \mathbb{F}_2^\ell \to \mathbb{F}_2^m$ for a t-sparse matrix $\mathbf{A}$, that is, where each row of $\mathbf{A}$ has exactly t non-zero entries:

$$g_{\mathbf{A}}(\mathbf{s}, \mathbf{r}) = \mathbf{A}\mathbf{s} + \mathsf{BinSamp}_p(\mathbf{r}) \bmod 2,$$

where $\mathsf{BinSamp}_p(\mathbf{r})$ outputs a vector $\mathbf{e} \in \mathbb{F}_2^m$ such that if $\mathbf{r}$ is a uniformly random vector, then each entry of $\mathbf{e}$ is an independent Bernoulli random variable with parameter p. If $\mathsf{BinSamp}_p(\mathbf{r})$ can be implemented as an NC^0 function, we will have a polynomial-stretch PRG computable in NC^0 from the SparseLPN assumption, and we would be done.

[3]More precisely, their construction can be adapted to work for constant sparsity t, noise rate $o(1/n^{\delta})$, and $m = n^{1+(t/2-1)(1-\delta)}$ samples. In Appendix A, we provide a summary of the existing cryptanalysis on SparseLPN, and we explain why this separation between parameter regimes may be inherent.

[4]We remark that it may be possible to also instantiate the $i\mathcal{O}$ construction assuming *sparse* LPN over $\mathbb{Z}_p$ instead of standard FieldLPN, so that our construction would rely on just DLIN as well as the SparseLPN assumption over both $\mathbb{Z}_2$ and large fields of prime order. For this variant of SparseLPN over $\mathbb{Z}_p$, we need sparsity $t = \omega(1)$ to allow for an arbitrary polynomial number of samples. Since the sub-exponential SparseLPN assumption over $\mathbb{Z}_p$ does not hold with all but *sub-exponential* probability over the randomness of uniform sparse $\mathbf{A}$ (for $t = n^{o(1)}$), this would require checking that the PPE construction by [JLS22] can be made compatible with our use of FE combiners. For simplicity, we do not pursue this generalization in the current version.

This turns out to be impossible. By standard results in the analysis of Boolean functions, any function $f : \{0,1\}^n \to \{0,1\}$ that outputs a sample from $\mathsf{Bern}(\eta)$ given a uniformly random input $\mathbf{r} \leftarrow \{0,1\}^n$ must have locality $\Omega(\log 1/\eta)$; in particular, if $\eta = o(1)$, as is needed for polynomial stretch, the locality is $\omega(1)$, which is insufficient in the construction of [JLS22].

Our Idea. In the outline above, there is no need to choose $\mathbf{r}$ from the uniform distribution; indeed, the distribution of $\mathbf{r}$ can be arbitrary, it turns out, subject to three constraints:

(a) $\mathbf{r}$ shouldn't be too long;

(b) expanding $\mathbf{r}$ into the Bernoulli $\mathbf{e}$ should be doable with a degree-$O(1)$ polynomial over $\mathbb{Z}$; and

(c) $\mathbf{r}$ can be sampled by a circuit whose size is sublinear in the PRG output length.

We note in passing that if it were for conditions (a) and (b) alone, one way to come up with such a distribution of $\mathbf{r}$ is to start with a Bernoulli $\mathbf{e}$ and to compress it using low-rank matrices, much the same way as [JLS21, JLS22]. Multiplying out the low-rank matrices ensures that expanding $\mathbf{r}$ to $\mathbf{e}$ can be done in degree 2. Assuming sub-exponential LWE in addition, we could recover the $i\mathcal{O}$ result [GKP$^+$13, LPST16].

To avoid the need for LWE and the bootstrapping results of [GKP$^+$13, LPST16], we instead *implicitly* sample $\mathbf{e}$ by sampling the list of entries where it is nonzero (which will be sublinear in the length of $\mathbf{e}$), and then directly construct the compressed representation using low-rank matrices as in [JLS21, JLS22]. Doing this with a sublinear-time RAM program is relatively straightforward, but some care is needed to show that this can be implemented as a sublinear-size circuit.

More generally, we abstract away our needs from sparse LPN into two separate objects that are both weaker than the existence of a polynomial-stretch PRG in NC^0.

Theorem 2 (Informal Version of Corollary 7.10, following [JLS22])**.** *There exists an $i\mathcal{O}$ scheme under assumptions (1) and (2) as well as the existence of*

(a) any "structured-seed" polynomial-stretch Boolean PRG computable by degree-$O(1)$ polynomials over $\mathbb{Z}$ with an arbitrarily small polynomial locality (see Section 2 and Definition 8 for a formal definition); and

(b) any linear stretch Boolean PRG in NC^0.

Both the $\mathsf{SparseLPN}$ assumption as well as the existence of Boolean PRGs in NC^0 imply the existence of both objects (a) and (b) above. This theorem is thus a common generalization of both [JLS22] and our result, a fact that we hope will be useful in further constructions of $i\mathcal{O}$ from fewer and/or simpler assumptions.

1.1 Consequences

A rich line of work initiated by [SW14] has shown a plethora of applications of $i\mathcal{O}$ to other problems in cryptography. As a result, our construction implies instantiations of these cryptographic primitives from sets of assumptions that were not previously known, in particular the *sub-exponential* hardness of assumptions (1) and (2) and the $\mathsf{SparseLPN}$ assumption. We list some of these below, following [JLS22].

- Fully homomorphic encryption (FHE), noting that bilinear DLIN implies a perfectly rerandomizable encryption scheme [CLTV15].
- Adaptively sound perfectly zero-knowledge succinct non-interactive argument (SNARG) system for any NP language in the CRS model, with CRS length $\mathsf{poly}(\lambda + |C|)$ where C is the NP verification circuit [SW14, WW24]. (The result by [WW24] holds assuming $i\mathcal{O}$ for $\mathsf{P/poly}$ and the hardness of the discrete logarithm problem over some prime-order group, which is immediate from DLIN in bilinear prime-order groups.)
- Public-key functional encryption for Turing machines that is fully succinct and adaptively secure against unbounded collusions [AS16]. Full succinctness means that the runtime of encrypting an input $\mathbf{x} \in \{0,1\}^*$ is simply $\mathsf{poly}(\lambda, |\mathbf{x}|)$, independent of the size and the runtime of the Turing machine being evaluated on $\mathbf{x}$.
- Witness encryption for any NP language, as a special case of $i\mathcal{O}$ for $\mathsf{P/poly}$.
- Secret sharing for any monotone function in NP [KNY17].
- Multiparty non-interactive key exchange in the plain model (without trusted setup) [BZ14, KRS15].
- Sender deniable encryption [SW14], and fully deniable interactive encryption [CPP20].
- Constant-round concurrent zero-knowledge proofs for any NP language [CLP15].

In addition, assuming the *polynomial* hardness of assumptions (1) and (2) and the $\mathsf{SparseLPN}$ assumption (over the explicit distribution of sparse $\mathbf{A}$ matrices given by [AK23]; see Section 3.4), we obtain the following (by polynomial hardness, we mean that p.p.t. adversaries in any of the assumptions achieve negligible advantage):

- Public-key functional encryption for polynomial-size circuits that is *adaptively* secure against unbounded collusions and fully succinct, i.e., the runtime of encrypting an input $\mathbf{x}$ is independent of the size of the circuit being evaluated on $\mathbf{x}$ [GS16, LM16, ABSV15, KNTY19]. This follows from our construction in Section 7.2 of PKFE for polynomial-size circuits that is selectively secure with a single key and weakly succinct.
- As a special case of the above, attribute-based encryption for polynomial-size circuits.
- Hardness of PPAD [AKV04, BPR15, GPS16, HY20, KS20].

Organization of the Paper. In Section 2, we provide a technical outline of our construction. In Section 3, we set up some notation and formally introduce our LPN and $\mathsf{SparseLPN}$ assumptions. In Section 4, we give the construction of structured-seed PRGs from $\mathsf{SparseLPN}$ over $\mathbb{Z}_2$; this is one of the main modifications we make to the construction of [JLS22]. In Sections 5, 6, and 7.2, we closely follow the constructions and analysis by [JLS22] of $\mathsf{ARE}, \mathsf{PRE}, \mathsf{PKFE}$ respectively, while making minor modifications as necessary to accommodate our notions of structured-seed PRGs and "combiner-friendliness". Finally, in Section 7.4, we make a white-box modification to the SKFE combiner by [JMS20] to bootstrap this to sub-exponentially secure SKFE, which can then be bootstrapped to $i\mathcal{O}$ [KNT22].

2 Technical Overview

Our starting point is the construction of $i\mathcal{O}$ from [JLS22]. Most of our construction follows exactly the same template as in [JLS22], with the main exception being that we replace the polynomial-stretch PRG in NC^0 with weaker forms of pseudorandomness. Our main idea can be split into two:

1. The requirement of a polynomial-stretch PRG in NC^0, as needed in framework of [JLS22], can be replaced with two weaker objects: (a) a *linear*-stretch PRG in NC^0, and (b) a polynomial-stretch *structured-seed PRG* (SPRG), which we define shortly.
2. We can build both such PRGs from the assumption that sparse LPN holds over $\mathbb{Z}_2$ for some fixed constant sparsity, some fixed polynomial number of samples, and some fixed inverse polynomial error rate. A linear-stretch PRG in NC^0 from sparse LPN directly comes from [AIK08]. We show how to construct a polynomial-stretch SPRG from sparse LPN later in this section.

Informally (ignoring some technicalities), the definition of an SPRG with polynomial stretch is as follows. We say G is an SPRG (with polynomial stretch and outputs in $\{0,1\}^m$) if there is a randomized Boolean circuit SdSamp with the following properties:

1. Pseudorandomness over seeds from SdSamp, which says
$$\{G(\mathsf{seed}) \mid \mathsf{seed} \leftarrow \mathsf{SdSamp}(m)\} \approx_c U_m,$$
where U_m is the uniform distribution over $\{0,1\}^m$, and SdSamp is supported on $\{0,1\}^{m^{1-\Omega(1)}}$;
2. The randomized circuit SdSamp has size $m^{1-\Omega(1)}$; and
3. G can be written as a polynomial of total degree $d = O(1)$ over $\mathbb{Z}$ with locality $O(m^\tau)$ for arbitrarily small constant $\tau > 0$.

This relaxes the notion of a PRG in NC^0 with polynomial stretch in two main ways: (1) the seed does not have to be uniformly random (just sampleable with sublinear efficiency), and (2) the locality can be an arbitrarily small polynomial in m, as long as the *degree* of the polynomial over $\mathbb{Z}$ is bounded by a constant $d = O(1)$. We emphasize that our degree condition is over $\mathbb{Z}$, but we require pseudorandomness over $\{0,1\}^m$.

Comparison with Structured-Seed PRGs in [JLS21]. We note that the $i\mathcal{O}$ construction of [JLS21] relies on another notion of structured-seed PRG that is incomparable to ours. Most importantly, we require SdSamp to have circuit size $m^{1-\Omega(1)}$, while [JLS21] imposes no such requirement. This is because [JLS21] ultimately constructs sublinear size-succinct FE, which they then bootstrap to sublinear time-succinct FE assuming LWE [GKP+13, LPST16] before finally bootstrapping to $i\mathcal{O}$ [AJ15, BV18]. We do not rely on LWE, so SdSamp must itself have an efficient circuit.

Secondly, [JLS21] allows SdSamp to generate a public and a private seed, and requires that decompression be degree 2 in the private seed and degree $O(1)$ in the public seed. In our case, it suffices for decompression to have degree $O(1)$, so we do not need to work with a public seed. In the other direction, it will be important to us for technical reasons that the locality of G can be bounded by an arbitrarily small polynomial in its output length, whereas [JLS21] does not require any such restriction.

Finally, we construct SPRG assuming only $\mathsf{SparseLPN}$ over $\mathbb{Z}_2$. It can also be trivially instantiated from a (polynomial-stretch) PRG in NC^0. In contrast, the [JLS21] construction of their notion of structured-seed PRG requires *both* a (polynomial-stretch) PRG in NC^0 and LPN over $\mathbb{Z}_p$.

2.1 Weakening the Polynomial-Stretch PRG in NC^0 in [JLS22]

Before describing why these weaker forms of pseudorandomness are sufficient to replace the polynomial-stretch PRG in NC^0 in [JLS22], we briefly summarize the overall template in [JLS22].

The starting point in [JLS22] is partially-hiding functional encryption (PHFE) [GVW15], which can be built from the DLIN assumption on symmetric bilinear groups of prime order [JLS19, GJLS21, Wee20]. This gives a special form of functional encryption (FE), where function keys can be given to functions that are degree-2 (over $\mathbb{Z}_p$) over the secret input SI, but allowed to be degree-$O(1)$ (or more generally, NC^1) over a *public* input PI that the FE scheme does not need to hide. If one could turn this into a sublinear time-succinct FE scheme for all polynomial size circuits (with sub-exponential security), then using known bootstrapping results [BV18, AJ15, KNT22], one gets $i\mathcal{O}$. By *time-succinct*, we mean that the time to generate a ciphertext (or more accurately, the size of the circuit generating the ciphertext) should be sublinear in the size of the circuit given in the function keys. More precisely, to handle function keys for circuits $C : \{0,1\}^n \to \{0,1\}^m$ of size s, we require the time to compute $\mathsf{FE.Enc}$ (more accurately, the circuit size) to be $s^{1-\Omega(1)} \cdot \mathsf{poly}(n, \lambda)$. This is a stronger notion than *size-succinct* FE, which only requires that the size of the ciphertext (i.e., output length of $\mathsf{FE.Enc}$) be sublinear, with no sublinear constraint on the time needed to generate the ciphertext. For size-succinct FE, we only know how to get $i\mathcal{O}$ from additionally assuming the learning with errors assumption (LWE) [LPST16, GKP+13], which we do not want to rely on.

Jain, Lin, and Sahai [JLS22] then define a cryptographic object called a Preprocessed Randomized Encoding (PRE), which exactly converts the PHFE as described above into a time-succinct FE for all polynomial size circuits, resulting in a construction of $i\mathcal{O}$. Roughly speaking, PRE splits up the computation of a given circuit C on an input $\mathbf{x}$ into 2 steps: (a) a preprocessing step, that needs to be time-succinct, generating $(\mathsf{PI}, \mathsf{SI})$ from $\mathbf{x}$, and (b) an encoding step, which is degree $(O(1), 2)$ in $(\mathsf{PI}, \mathsf{SI})$ (i.e., total degree $O(1)$ in PI and total degree 2 in SI), that outputs a randomized encoding of $C(\mathbf{x})$. One can directly plug this into a PHFE to get time-succinct FE for all polynomial-size circuits (assuming the PHFE encryption time is linear, which it is).

To build PRE, which is sufficient for $i\mathcal{O}$ as explained above, [JLS22] build it modularly from two objects that they construct: a Preprocessed Polynomial Encoding (PPE), and an Amortized Randomized Encoding (ARE). Roughly speaking, PPE preprocesses any $\mathbf{x}$ into $(\mathsf{PI}, \mathsf{SI})$ in such a way that any degree-$O(1)$ computation on $\mathbf{x}$ (over $\mathbb{Z}_p$) is turned into a degree-$(O(1), 2)$ computation in $(\mathsf{PI}, \mathsf{SI})$. Crucially, this preprocessing step is time-succinct. [JLS22] show how to build PPE directly from (standard) LPN over $\mathbb{Z}_p$ with polynomially many samples and any inverse polynomial error rate, by constructing a special-purpose homomorphic encryption scheme tailored to constant-degree computations.

Their last missing piece, ARE, generates a (binary) randomized encoding of computing a circuit C on input $\mathbf{x}$ (e.g., using Yao's garbled circuits [Yao86]) in such a way that the encoding algorithm has constant locality, i.e., is in NC^0. Since the encoding algorithm is in NC^0, taking the multilinear representation over $\mathbb{Z}$, this directly becomes a degree-$O(1)$ computation over $\mathbb{Z}$ and hence $\mathbb{Z}_p$. Plugging such an ARE into PPE directly gives PRE, as desired. (For simplicity, we gloss over the amortization constraint here, which allows their composition with PPE to be time-succinct.)

One important property here is that Yao's garbled circuits needs pseudorandomness in two ways:

1. For computing the garbled tables in Yao's garbled circuits, the encoding and decoding requires computing a length-doubling PRG to preserve pseudorandomness of unused entries in the garbled tables. To retain $O(1)$-degree over $\mathbb{Z}$ in the encoding, [JLS22] assumes the existence of a polynomial-stretch PRG in NC^0, which, in particular, gives a linear-stretch PRG in NC^0.

2. The wire labels in the garbled circuit need to be random and hidden. Since PPE does not guarantee any function privacy, we need to hide this randomness in the input to PPE. To retain time-succinctness of PPE preprocessing, the randomized encoding uses a polynomial-stretch PRG by including the seed as part of the input to PPE. To retain $O(1)$-degree over $\mathbb{Z}$, [JLS22] assumes the existence of a polynomial-stretch PRG in NC^0 so that the encoding is still $O(1)$-degree over $\mathbb{Z}$. (For technical reasons, for time-succinctness, when using the PPE, we need the number of monomials to be arbitrarily close to linear in the output length of the randomized encoding.)

This is exactly the place where we can use weaker forms of pseudorandomness than a polynomial-stretch PRG in NC^0. To solve Item 1, we can directly use a linear-stretch PRG in NC^0, and to solve Item 2, we can exactly use our notion of a structured-seed PRG. As explained above, being degree $O(1)$ over $\mathbb{Z}$ is sufficient, and to retain time-succinctness, we need to make sure that PRE, and therefore ARE, can sample these seeds in a time-succinct way. This is exactly the constraint we have on seed sampling in an SPRG.

We briefly explain why an SPRG alone (i.e., without the standard linear-stretch PRG) is insufficient. For Yao's garbled circuit evaluation, we need composability of the linear-stretch PRG, as we feed the outputs of the the linear-stretch PRG as an input to the PRG in the next gate of the circuit. Unfortunately, having a structured seed ruins this composability, as sampling the seed from (unstructured) PRG output need not be degree $O(1)$. Alternatively, for the polynomial-stretch PRG, we only need to evaluate it once, so there is no need for composability; the seed sampling can just happen once at the PRE preprocessing level, as long as it is time-succinct.

More generally, taking a step back, we view our definition of SPRG as naturally fitting into the [JLS22] paradigm. Just as in [JLS22], we separate the computation (in this case, PRG evaluation) into 2 steps: (1) an efficient preprocessing step (i.e., in a time-succinct way), and (2) computing degree-$O(1)$ polynomials on top of the preprocessing. We exactly harness this flexibility that is built into the [JLS22] framework to build our SPRG.

We summarize our construction and how it compares with the construction of [JLS22] in Figure 1.

2.2 Our SPRG Construction from Sparse LPN

In this section, we explain how we construct the two weaker pseudorandom objects that we need, namely a linear-stretch PRG in NC^0 and a polynomial-stretch structured-seed PRG. It has been shown by [AIK08] that the $\mathsf{SparseLPN}$ assumption implies the existence of a linear-stretch PRG in NC^0. However, as the authors of [AIK08] mention, their techniques do not yield a PRG in NC^0 with superlinear stretch, let alone polynomial stretch.

As explained in Section 2.1, we observe that the polynomial-stretch PRG would only be needed to generate pseudorandom bits to use instead of randomness for Yao's garbled circuits [Yao86]; it does not matter whether the seed is uniformly random or has some structure. $\mathsf{SparseLPN}$ appears

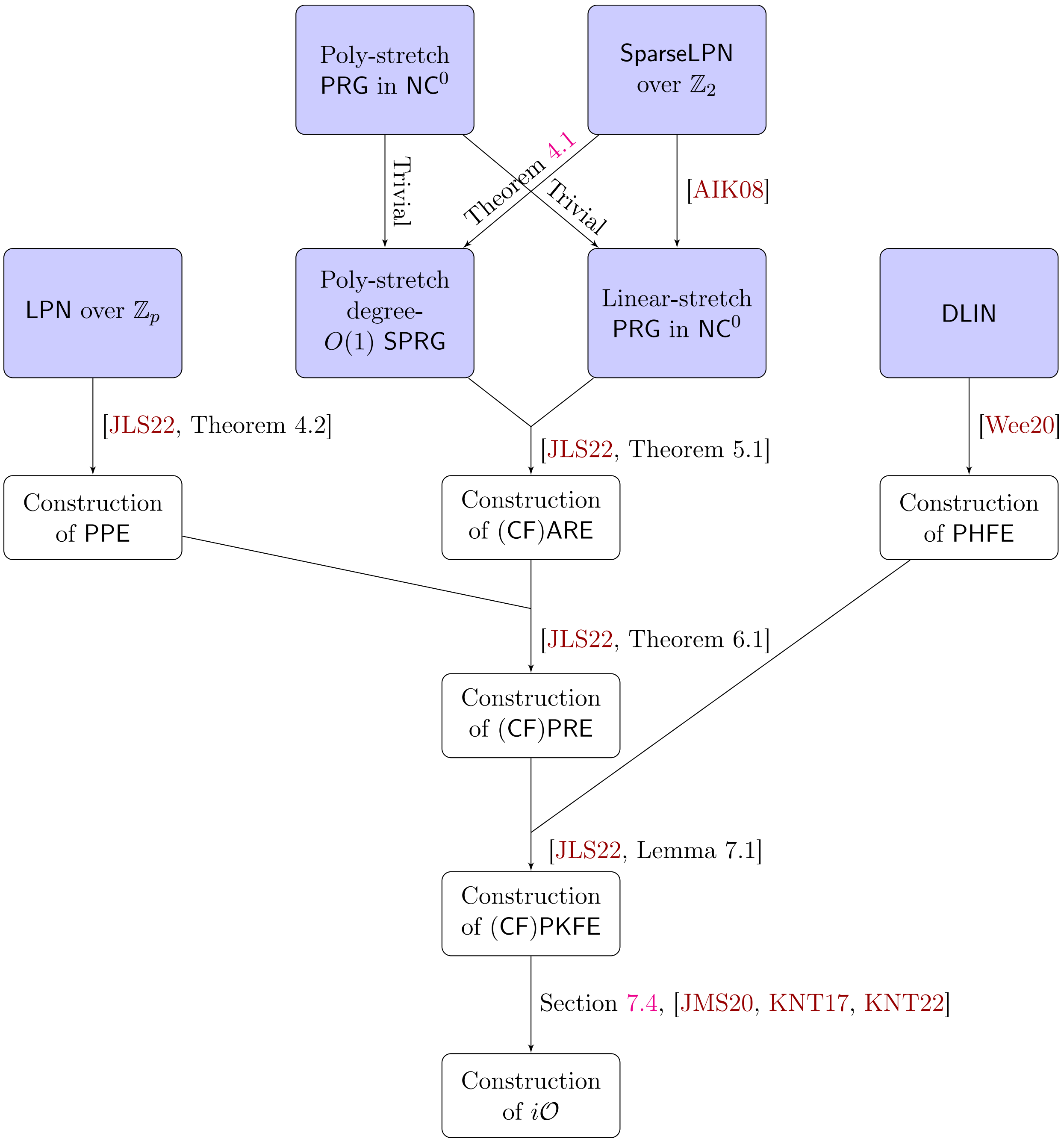

Figure 1: Flowchart depicting our technical outline to get to $i\mathcal{O}$, following the framework of [JLS22]. The prefix CF stands for "combiner-friendly", and we inherit the notions of PPE (Preprocessed Polynomial Encoding), ARE (Amortized Randomized Encoding), PRE (Preprocessed Randomized Encoding), PHFE (Partially Hiding Functional Encryption), and PKFE (Public-Key Functional Encryption) from [JLS22]. Our observation is that [JLS22] ultimately needs two abstractions from their PRG in NC^0, namely a poly-stretch degree-$O(1)$ structured-seed PRG, and a linear-stretch PRG in NC^0. We show in this work that these two abstractions can also be instantiated assuming SparseLPN, thus providing an alternate construction of $i\mathcal{O}$. Note that everything here needs to be sub-exponentially secure to get to $i\mathcal{O}$. We provide more details on the bootstrapping involved in the final arrow in Figure 2.

like a natural candidate for this functionality: given a secret vector $\mathbf{s} \in \mathbb{Z}_2^n$ and a sparse (hence, structured) error vector $\mathbf{e} \in \mathbb{Z}_2^m$, it expands $\mathbf{s}$ to $g_{\mathbf{A}}(\mathbf{s}, \mathbf{e}) := \mathbf{A}\mathbf{s} \oplus \mathbf{e} \in \mathbb{Z}_2^m$, where $\{g_{\mathbf{A}}\}_{\mathbf{A}}$ would comprise a candidate SPRG family. The map $\mathbf{s} \mapsto \mathbf{A}\mathbf{s}$ has locality $O(1)$ due to the row sparsity of $\mathbf{A}$, and hence it has degree $O(1)$ over $\mathbb{Z}$. (We remark that $\mathbf{s} \mapsto \mathbf{A}\mathbf{s}$ is linear over $\mathbb{Z}_2$, but this in and of itself does not imply anything nontrivial about its degree over $\mathbb{Z}$.) Similarly, the map $(\mathbf{A}\mathbf{s}, \mathbf{e}) \mapsto \mathbf{A}\mathbf{s} \oplus \mathbf{e}$ has locality 2 and hence also has degree $O(1)$ over $\mathbb{Z}$. It follows that $g_{\mathbf{A}}$ has locality $O(1)$ (and hence degree $O(1)$ over $\mathbb{Z}$).

However, there remains the conundrum of how the error vector should be handled. If the error vector is included as-is in the structured seed, the resulting "SPRG" would map $m+n$ to m bits and hence not even be expanding. Instead, we could try to sample $\mathbf{e}$ using a polynomially smaller number of uniformly random bits. However, by standard facts from the analysis of Boolean functions, it is impossible to generate samples from $\mathsf{Bern}(\eta)$ for $\eta = o(1)$ from any $O(1)$-degree or NC^0 function over $\mathbb{Z}$ from uniformly random bits, regardless of locality [O'D14, e.g., Exercise 1.11].

Instead, we approach this problem by compressing the error vector into some $\widetilde{\mathbf{e}} \in \{0,1\}^{m'}$, where $m' = m^{1-\Omega(1)}$, such that the decompression map $\widetilde{\mathbf{e}} \mapsto \mathbf{e}$ has degree $O(1)$ over $\mathbb{Z}$. Since $\mathbf{e}$ is polynomially sparse, we can use the beautiful idea given by [JLS21, JLS22] of using low-rank matrix decompositions. We now provide an overview of this technique.

Given the $\mathsf{SparseLPN}$ noise rate $\eta = m^{-\Omega(1)}$, we can set parameters ℓ, L such that $L \ll 1/\eta$ is a small polynomial in m and $\ell \cdot L^2 = m$. We can accordingly reorganize the entries of $\mathbf{e} \in \{0,1\}^m$ into ℓ matrices $\mathbf{M}_1, \ldots, \mathbf{M}_\ell \in \{0,1\}^{L \times L} \subseteq \mathbb{Z}^{L \times L}$. For each i, the entries in $\mathbf{M}_i$ are independently sampled Bernoulli random variables with probability η. Hence the expected number of nonzero entries in $\mathbf{M}_i$ is $L^2\eta \ll L$. In particular, the rank of $\mathbf{M}_i$ over $\mathbb{Z}$ will be at most $L' = O(L^2\eta)$. We can hence write each $\mathbf{M}_i = \mathbf{U}_i \cdot \mathbf{V}_i$ where $\mathbf{U}_i, \mathbf{V}_i^\top \in \mathbb{Z}^{L \times L'}$ (in fact, $\{0,1\}^{L \times L'}$), and finally output $\{(\mathbf{U}_i, \mathbf{V}_i)\}_{i \in [\ell]}$ as the compressed error $\widetilde{\mathbf{e}}$. Decompression can be done in degree 2 over $\mathbb{Z}$, and the size of $\widetilde{\mathbf{e}}$ is $O(\ell \cdot L \cdot L') = O(\ell \cdot L^3\eta) = O(m^{1-\Omega(1)})$.

This would appear to immediately give us a SPRG from $\mathsf{SparseLPN}$. The problem is that although this structured seed is polynomially smaller than m, the time taken to construct it could be polynomial in m (in particular, it would have to be at least $\Omega(m)$ just to read $\mathbf{e}$). Instantiating the [JLS22] scheme with this construction would yield a sublinear *size*-succinct FE scheme, where the ciphertexts are succinct but we do not impose a constraint on the time taken to generate them. This would have to first be bootstrapped to sublinear *time*-succinct FE [GKP+13, LPST16] assuming LWE, which can then be bootstrapped to $i\mathcal{O}$ [BV18, AJ15, KNT17, KNT22]. Ideally, we would want to construct $i\mathcal{O}$ without the need for LWE and the bootstrapping results of [GKP+13, LPST16].

To do this, we need to sample $\widetilde{\mathbf{e}}$ with a circuit of size $m^{1-\Omega(1)}$. In particular, we cannot even explicitly sample $\mathbf{e} \sim \mathsf{Bern}(\eta)^m$. Instead, we implicitly sample it as follows: we first sample its Hamming weight which will be $\mathsf{wt} = O(m\eta) = O(m^{1-\Omega(1)})$ with all but sub-exponentially small probability. Then, rather than sampling $\mathbf{e}$ in its entirety, we simply sample the list of locations where $\mathbf{e}$ is nonzero. This information is sufficient to directly construct the matrices $\{\mathbf{U}_i, \mathbf{V}_i\}_{i \in [\ell]}$. This will be doable in time $\mathsf{wt} \cdot \mathsf{poly}(\log m) = O(m^{1-\Omega(1)})$.

As in [JLS22], we need to take particular care to ensure that our algorithm is implementable with a sublinear-size *circuit*, rather than a RAM program (this is because the bootstrapping results of [BV18, AJ15, KNT17, KNT22] impose this requirement on the encryption of the FE, which is where our SPRG will sample its structured seed). Fortunately, this is not too difficult and follows using similar approaches to those used by [JLS22]; at a high level, our algorithm can be decomposed into sorting steps [AKS83] and $O(1)$-tape Turing machine computations [PF79], both of which are known to be achievable using circuits with low overheads.

We make some remarks comparing our use of the techniques by [JLS22] for compressing with low-rank matrices and doing this with a sublinear-time circuit, with how they were originally used in [JLS22]. Firstly, while we use these techniques to instantiate the SPRG (which is intended to replace the PRG in NC^0 used by [JLS22]), these techniques were originally used by [JLS22] to instantiate Preprocessed Polynomial Encodings (PPE) assuming LPN over $\mathbb{Z}_p$. We make black-box use of their PPE construction, and hence our construction implicitly relies on these techniques in two different places: our SPRG and the PPE of [JLS22].

Secondly, our use of these techniques is simpler than that of [JLS22]. This is because the sparse vector that we need to compress has very little structure: its entries are simply i.i.d. Bernoulli random variables. On the other hand, the vector that [JLS22] needs to compress is also polynomially sparse but is highly structured; it contains information about how LPN errors propagate through homomorphic evaluations in the special-purpose homomorphic encryption scheme that they use. Keeping track of these errors requires much more careful bookkeeping than we need for our purposes. Although technically their construction also only requires the results of [PF79, AKS83], they crucially rely on the observation that these results can be used to efficiently make batched non-adaptive RAM queries to a database [JLS22, Lemma 4.6], whereas our construction does not need such complex functionality.

2.3 Our Use of FE Combiners

Let $\mathsf{SparseMat}(t, n, m)$ denote the set of matrices $\mathbf{A} \in \mathbb{Z}_2^{m\times n}$ whose rows all have sparsity exactly t. An "ideal" version of the $\mathsf{SparseLPN}$ assumption would say we have the sub-exponential indistinguishability

$$\begin{aligned}&\{(\mathbf{A}, \mathbf{u} = \mathbf{A}\mathbf{s} + \mathbf{e} \mod 2) \mid \mathbf{A} \leftarrow \mathsf{SparseMat}(t, n, m), \mathbf{s} \leftarrow \mathbb{Z}_2^n, \mathbf{e} \leftarrow \mathsf{Bern}(\eta)^m\} \\ \approx_c &\{(\mathbf{A}, \mathbf{u}) \mid \mathbf{A} \leftarrow \mathsf{SparseMat}(t, n, m), \mathbf{u} \leftarrow \mathbb{Z}_2^m\}.\end{aligned}$$

However, for $t = O(1)$, this is *false*. In particular, with at least $1/\mathsf{poly}(n)$ probability, there exist two identical rows of $\mathbf{A}$, implying there is an (efficiently computable) vector of sparsity 2 in the left kernel of $\mathbf{A}$. By left-multiplying this vector with $\mathbf{u}$, this can be used to break indistinguishability when $\eta = o(1)$.

For $\mathbf{A}$ that do not have such sparse vectors in the left kernel (more formally, when the dual distance of $\mathbf{A}$ is large; see Appendix A), there are no known attacks that work with better than sub-exponential advantage. For the negligible (but not sub-exponential) security regime, [AK23] gives an efficient sampler for sparse matrices $\mathbf{A}$ for which negligible security is plausible. Therefore, there is no issue with the above-mentioned template to instantiate negligible-secure sublinear-time (single-key) public-key FE. This allows for a simpler construction of public-key FE and its downstream applications in the negligible (but not sub-exponential) security regime.

However, to bootstrap to $i\mathcal{O}$, one needs sub-exponential security. Unfortunately, there is no known algorithm to efficiently sample these good $\mathbf{A}$ with plausible sub-exponential security. Thus, our sparse LPN assumption has the flavor that there exist *not necessarily efficiently sampleable* distributions $\mathsf{GoodSparseMat}(t, n, m)$ and $\mathsf{BadSparseMat}(t, n, m)$ such that $\mathsf{SparseMat}(t, n, m)$ can be written as a mixture of $\mathsf{GoodSparseMat}(t, n, m)$ and $\mathsf{BadSparseMat}(t, n, m)$, with weight at least $\mu = \Omega(1)$ on $\mathsf{GoodSparseMat}(t, n, m)$, and that (sub-exponential) indistinguishability holds with respect to $\mathbf{A} \leftarrow \mathsf{GoodSparseMat}(t, n, m)$. The issue is that we only know how to efficiently sample from $\mathsf{SparseMat}(t, n, m)$, but we can claim sub-exponential security only when sampling from $\mathsf{GoodSparseMat}(t, n, m)$.

To address this problem, there are two natural approaches:

Idea 1 The first idea would be to combine SPRG outputs using the standard XOR approach: given sparse LPN matrices $\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_t$, we could instead consider a "combined SPRG" that independently calls $\mathsf{SPRG.SdSamp}$ t times to obtain (structured) seeds $\mathbf{r}_1, \ldots, \mathbf{r}_t$, then at the decompression stage computes $\bigoplus_{i=1}^{t} \mathsf{SPRG.Eval}(\mathbf{A}_i, \mathbf{r}_i)$. This will be secure as long as at least one of the sparse LPN matrices $\mathbf{A}_1, \ldots, \mathbf{A}_t$ is "good."

The problem we now face is that if we want one of the sparse LPN matrices to be good with all but a sub-exponentially small probability, t will have to be $\lambda^{\Omega(1)}$, resulting in the degree of the SPRG decompression growing too fast.

On the other hand, if $t = O(1)$, then the SPRG decompression will still be constant degree. Jumping ahead, this observation will turn out to be helpful for us, even though this idea on its own will not directly achieve sub-exponential security.

Idea 2 Alternatively, we could instantiate polynomially many copies of our final FE scheme and plug these into an unconditional FE combiner [ABJ+19, JMS20].

The central problem with this approach is that the combiner of [JMS20] does not directly preserve sublinear time-succinctness of the FE. In their construction, their first step is to use $O(1)$-*nested* FE candidates [ABJ+19], which is a direct construction of an FE combiner for $O(1)$ candidates. Informally, for a constant $B = O(1)$, B-nesting FE candidates $\{\mathsf{FE}_i\}_{i \in [B]}$ refers to the following construction. To encrypt, set

$$\mathsf{FENest.Enc}(\mathbf{x}) = \mathsf{FE}_B.\mathsf{Enc}(\cdots(\mathsf{FE}_2.\mathsf{Enc}(\mathsf{FE}_1.\mathsf{Enc}(\mathbf{x})))\cdots).$$

To produce a function key $\mathsf{FENest.KeyGen}(\mathsf{MSK}, C)$, one releases SK_B, defined by the following iterative process: $\mathsf{SK}_1 = \mathsf{FE}_1.\mathsf{KeyGen}(\mathsf{MSK}_1, C)$, and for $i \in [B-1]$, $G_i = \mathsf{FE}_i.\mathsf{Dec}(\mathsf{SK}_i, \cdot)$, where $\mathsf{SK}_{i+1} = \mathsf{FE}_{i+1}.\mathsf{KeyGen}(\mathsf{MSK}_{i+1}, G_i)$. Since there is no constraint on the size of the decryption circuits, there is no guarantee that we get time succinctness.

However, it turns out that we can integrate these two ideas by making white-box use of the results of [JMS20]. Let's first recall the ideas underlying their combiner. Given candidate FE schemes $\{\mathsf{FE}_i\}_{i \in [t]}$, the combiner of [JMS20] comprises two steps:

1. They first use a naive 3-nesting-based approach to construct FE schemes $\{\mathsf{FE}_{i_1,i_2,i_3}\}_{i_1,i_2,i_3 \in [t]}$. The only structure they need here is the existence of an index $i^* \in [t]$ such that the schemes $\{\mathsf{FE}_{i^*,i_2,i_3}\}, \{\mathsf{FE}_{i_1,i^*,i_3}\}, \{\mathsf{FE}_{i_1,i_2,i^*}\}$ are all secure. However, this is the part of their construction that does not preserve sublinear time-succinctness.

2. They then use these nested FE candidates to compute a transcript of an "input-local" semi-honest t-party secure multi-party computation (MPC) protocol, with up to $t-1$ corrupted parties. Roughly speaking, for $i \in [t]$, each party P_i gets an XOR secret share $\mathbf{x}_i$ of the input $\mathbf{x}$, and they together run the MPC protocol for outputting $C(\mathbf{x}_1 \oplus \cdots \oplus \mathbf{x}_t)$ for a given circuit C. Each bit of the transcript depends only on 3 parties (and their correlated randomness), and they use the nested FE candidates to compute each bit of the transcript, from which the

MPC output can be recovered. As long as there is some $i^* \in [t]$ such that all 3-nested FE candidates that include i^* are secure, then they can argue security of the overall FE scheme. By instantiating this template with a special form of Yao's garbled circuits [Yao86, GS22, GIS18] and the [GMW87] MPC protocol, the efficiency can be made linear in the circuit size, which is sufficient to preserve time-succinctness. (They rely only on the existence of one-way functions, in particular by instantiating the correlated randomness model with PRFs.)

Another way to view this step is that it bootstraps the 3-nesting construction, which is secure provided at least one of 3 FE schemes is secure, to a FE scheme which is secure provided at least one of $\mathsf{poly}(\lambda)$ FE schemes is secure. Crucially, this bootstrapping preserves succinctness.

Our observation is that Step 1 does not specifically need to use 3-nesting to construct the schemes $\{\mathsf{FE}_{i_1,i_2,i_3}\}$. Rather, all they need is:

1. the property that if at least one of $\mathsf{FE}_{i_1}, \mathsf{FE}_{i_2}, \mathsf{FE}_{i_3}$ is secure, then $\mathsf{FE}_{i_1,i_2,i_3}$ is secure; and

2. the property that the encryption of $\mathsf{FE}_{i_1,i_2,i_3}$ is sublinear time-succinct.

We formalize these properties through the notion of a *combiner-friendly* secret-key functional encryption scheme (CFSKFE). A CFSKFE scheme samples B common reference strings in a setup phase, and then uses all of the crs's in $\mathsf{KeyGen}, \mathsf{Enc}$ and Dec. If at least one of the crs's results in a sub-exponentially secure FE scheme, then the CFSKFE instantiated with these three crs's should also be sub-exponentially secure. Jumping ahead, we remark that for us, the crs's will be the $\mathsf{SparseLPN}$ $\mathbf{A}$ matrices and a good crs will be one sampled from $\mathsf{GoodSparseMat}$. Furthermore, setting $B = 3$ will suffice for us. To construct such a CFSKFE, we can simply use Idea 1 above, namely combine FE instances at the SPRG level, to instantiate the scheme $\mathsf{FE}_{i_1,i_2,i_3}$. Since there are only $O(1)$ many FE schemes being combined in this way, the degree of SPRG decompression will now remain $O(1)$.

Once we have such a CFSKFE, it can replace Step 1 in [JMS20], and can then be be bootstrapped to handle $\mathsf{poly}(\lambda)$ many crs's using Step 2 of [JMS20], while preserving sublinearity. Now we only need at least one of $\mathsf{poly}(\lambda)$ many crs's to be sub-exponentially secure, which will hold with all but sub-exponentially small probability.

To make this combiner statement modular and potentially useful to downstream works, in Theorem 7.7, we state a more general version of the FE combiner result given by [JMS20] that now preserves succinctness, as long as the underlying FE candidates have this "combiner-friendliness."

Since the construction of [JMS20] restricts attention to secret-key FE combiners, we ultimately obtain a (sub-exponentially) secure secret-key FE scheme that only supports single function queries. At a high level, our construction (like the construction of [JLS22]) does not support multiple function queries because the SPRG seeds would then be reused across multiple uses of Yao's garbled circuit construction [Yao86]. Finally, we bootstrap this construction to $i\mathcal{O}$ using results of [KNT17, KNT22]. We note that the other bootstrapping results we are aware of would not suffice for our purposes; the results of [BV18, AJ15] consider *public-key* FE, while the result of [BNPW20] considers secret-key FE but requires security with polynomially many function queries. It is plausible that the [JMS20] combiner can be directly made to work for public-key FE, but for simplicity, we use it as is.

We summarize our bootstrapping pipeline using combiners in Figure 2.

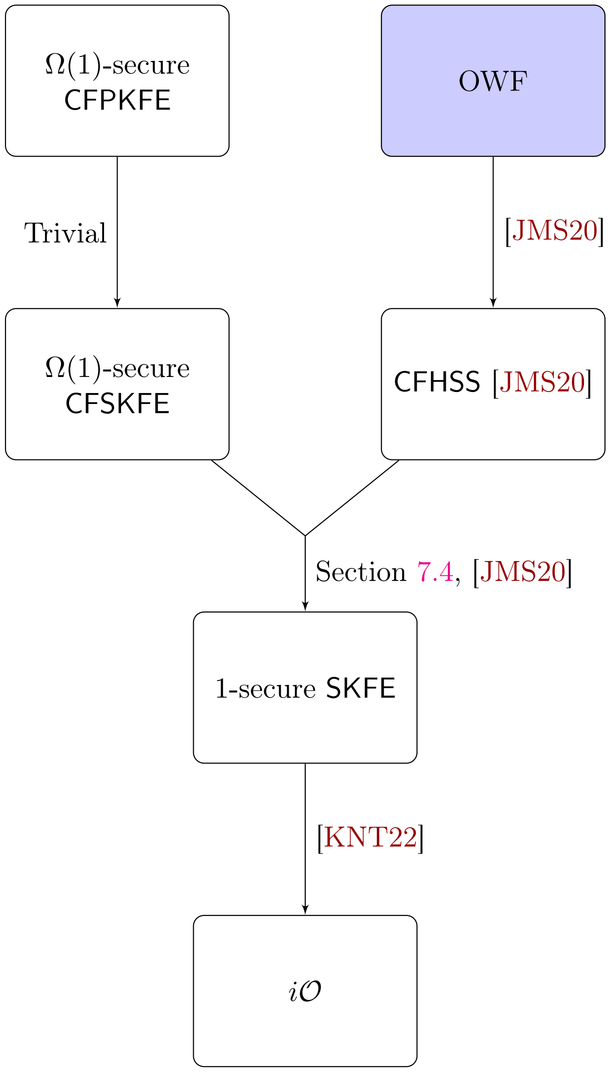

Figure 2: Flowchart depicting our method for bootstrapping combiner-friendly PKFE (public-key functional encryption) to $i\mathcal{O}$. We rely on a white-box modification of the SKFE (secret-key functional encryption) combiner constructed by [JMS20], to construct a single-key SKFE that is sub-exponentially secure with all but sub-exponentially small probability. This can then be bootstrapped to $i\mathcal{O}$ as shown by [KNT22]. Here, 1-secure (single-key) SKFE is simply (single-key) SKFE; we call it 1-secure just to compare to the other objects. Also, CFHSS refers to a combiner-friendly homomorphic secret sharing, as needed in the unconditional SKFE combiner [JMS20]. Note that everything here needs to be sub-exponentially secure to get to $i\mathcal{O}$. In the negligible but not sub-exponential regime, one can efficiently sample (plausibly) secure $\mathbf{A}$ for SparseLPN efficiently using [AK23], so one can go straight to secure PKFE without any consideration of combiners or combiner-friendliness, as in Figure 1.

3 Preliminaries

3.1 Notation

For any positive integer n, we let U_n denote the uniform distribution over strings in $\{0,1\}^n$. Let $\mathsf{Binom}(n,p)$ denote the binomial distribution that counts the number of successes in n independent Bernoulli trials with probability p. Wherever we work with polynomials in this paper, we assume they are represented as a list of monomial-coefficient pairs.

Throughout the paper, by a p.p.t. algorithm or adversary, we mean a non-uniform probabilistic polynomial time algorithm.

Definition 1 (Indistinguishability). *We say that two ensembles of distributions $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda\in\mathbb{N}}$ and $\mathcal{Y} = \{\mathcal{Y}_\lambda\}_{\lambda\in\mathbb{N}}$ are* indistinguishable *if for all p.p.t. adversaries $\mathcal{A}$, there exists a negligible function* negl *such that for all sufficiently large $\lambda \in \mathbb{N}$,*

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

Moreover, we say two ensembles $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda\in\mathbb{N}}$ and $\mathcal{Y} = \{\mathcal{Y}_\lambda\}_{\lambda\in\mathbb{N}}$ are sub-exponentially indistinguishable *if there exists a real number $c > 0$ such that for all p.p.t. adversaries $\mathcal{A}$,*

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

As short hand, we sometimes use the notation $\approx_c$ to denote indistinguishability, where the parameters are clear from context.

3.2 Locality and Degree

For finite sets A and B and a function $f : A^{n_1} \to B^{n_2}$, we will use the notation

$$f(\mathbf{x}) = (f_1(\mathbf{x}), \cdots, f_{n_2}(\mathbf{x})),$$

where for all $j \in [n_2]$, $f_j : A^{n_1} \to B$. For $\mathbf{x} \in A^{n_1}$ and a subset $S \subseteq [n_1]$, we will let $\mathbf{x}|_S \in A^{|S|}$ denote the restriction of $\mathbf{x}$ to indices in S.

Definition 2 (Locality). *For $n_1, n_2 \in \mathbb{N}$, finite sets A and B, and functions $f : A^{n_1} \to B^{n_2}$, we define the* locality *of f, $\mathsf{loc}(f)$, to be the maximum possible of input variables that any output coordinate depends on. More formally,*

$$\mathsf{loc}(f) := \max_{j\in[n_2]} \min \left\{ k \in \mathbb{N} : \exists S \subseteq [n_1], |S| = k, \exists \widehat{f_j} : A^{|S|} \to B \text{ s.t. } \forall \mathbf{x} \in A^{n_1}, f_j(\mathbf{x}) = \widehat{f_j}(\mathbf{x}|_S) \right\}.$$

By the union bound, we immediately have the following useful lemma.

Lemma 3.1. *For $n_1, n_2, n_3 \in \mathbb{N}$, finite sets A_1, A_2, A_3, and functions $f : A_1^{n_1} \to A_2^{n_2}$ and $g : A_2^{n_2} \to A_3^{n_3}$, we have*

$$\mathsf{loc}(g \circ f) \leq \mathsf{loc}(g) \cdot \mathsf{loc}(f).$$

Definition 3 (Degree). *For a multivariate polynomial $f : \mathbb{Z}^n \to \mathbb{Z}$, we let $\deg(f)$ denote the total degree of f. For a multi-output multivariate polynomial $f : \mathbb{Z}^{n_1} \to \mathbb{Z}^{n_2}$, we let*

$$\deg(f) := \max_{j\in[n_2]} \deg(f_j).$$

We recall a standard bound on the degree of a composition of polynomials.

Lemma 3.2. *For $n_1, n_2, n_3 \in \mathbb{N}$ and polynomials $f : \mathbb{Z}^{n_1} \to \mathbb{Z}^{n_2}$ and $g : \mathbb{Z}^{n_2} \to \mathbb{Z}^{n_3}$, we have*

$$\deg(g \circ f) \leq \deg(g) \cdot \deg(f).$$

Definition 4 (Definition 4.1 of [JLS22])**.** *For integers $n > d$, we say $\mathcal{Q}$ is a d-monomial pattern over n variables if $\mathcal{Q} = \{Q_1, \cdots, Q_m\}$, where for every $i \in [m]$, Q_i is a distinct subset of $[n]$ and $0 < |Q_i| \leq d$. For any input $\mathbf{x} \in \{0,1\}^n$ and subset $Q \subseteq [n]$, we define*

$$\mathsf{Mon}_Q(\mathbf{x}) := \prod_{i \in Q} x_i$$

to be the monomial in $\mathbf{x}$ corresponding to subset Q. That is, for any input $\mathbf{x}$, a d-monomial pattern $\mathcal{Q}$ over n variables defines a set of m distinct monomials in $\mathbf{x}$ of degree at most d.

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

We will use the notion of a d-monomial pattern when using Preprocessed Polynomial Encodings (PPE), as in [JLS22].

Lemma 3.3. *Let $f : \mathbb{Z}^n \to \mathbb{Z}$ be a polynomial with $d = \deg(f)$ and $\ell = \mathsf{loc}(f)$, where $d < \ell/2$. There exists $m \leq 1 + \ell^d$, a set of subsets $\mathcal{Q} = \{Q_1, \cdots, Q_m\}$, where $Q_i \subseteq [n]$ and $|Q_i| \leq d$, and a sequence of integers $(\zeta_i)_{i \in [m]}$ such that for all $\mathbf{x} \in \{0,1\}^n \subseteq \mathbb{Z}^n$,*

$$f(\mathbf{x}) = \sum_{i \in [m]} \zeta_i \cdot \mathsf{Mon}_{Q_i}(\mathbf{x}).$$

That is, f can be written as a linear combination of m multilinear monomials, where $m \leq 1 + \ell^d$.

Proof. Since $\mathsf{loc}(f) = \ell$, we can restrict the domain of f to $\mathbb{Z}^\ell$, and since we only need to preserve the values of f on $\{0,1\}^n$, we can restrict to multi-linear monomials, where the individual degree of any term is at most 1. Since $\deg(f) = d$, this means that no monomial has more than d terms. Therefore, since $d < \ell/2$, we can bound the number of monomials by

$$\sum_{i=0}^{d} \binom{\ell}{i} = 1 + \sum_{i=1}^{d} \binom{\ell}{i} \leq 1 + d\binom{\ell}{d} \leq 1 + \frac{\ell!}{(\ell-d)!(d-1)!} \leq 1 + \ell^d,$$

as desired. □

3.3 Pseudorandom Generators (PRGs)

In this section, we define the standard notion of a pseudorandom generator (PRG) family, but where the computational indistinguishability is required to hold with probability only $\Omega(1)$ over the choice of the function index.

Definition 5 (PRG Definition)**.** *A PRG family of stretch $\ell : \mathbb{N} \to \mathbb{N}$ consists of the following p.p.t. algorithms:*

- $\mathsf{IdSamp}(1^{n_{\mathsf{PRG}}}) \to I$: *The function index generation algorithm is randomized, and outputs a function index I.*

- $\mathsf{Eval}(I, \mathbf{x} \in \{0,1\}^{n_{\mathsf{PRG}}})$*: Deterministically outputs a string* $\mathbf{y} \in \{0,1\}^{\ell(n_{\mathsf{PRG}})}$.

***Efficiency**: We say that* PRG *is in* NC^0 *if for all* I *in the support of* $\mathsf{IdSamp}(1^{n_{\mathsf{PRG}}})$*, the mapping* $\mathsf{Eval}(I, \cdot) : \{0,1\}^{n_{\mathsf{PRG}}} \to \{0,1\}^{\ell(n_{\mathsf{PRG}})}$ *is in* NC^0*, i.e., each output bit depends on a constant number of input bits.*

***Security**: we say that* PRG *is (sub-exponentially)* $\mu(n_{\mathsf{PRG}})$*-secure if there exist (not necessarily efficiently sampleable) distributions* $\mathsf{GoodIdSamp}(1^{n_{\mathsf{PRG}}})$ *and* $\mathsf{BadIdSamp}(1^{n_{\mathsf{PRG}}})$ *such that both of the following hold:*

- *The following distributions are (sub-exponentially) indistinguishable:*

$$\left\{ (I, \mathsf{Eval}(I, \mathbf{x})) \;\middle|\; \begin{array}{c} I \leftarrow \mathsf{GoodIdSamp}(1^{n_{\mathsf{PRG}}}), \\ \mathbf{x} \leftarrow U_{n_{\mathsf{PRG}}} \end{array} \right\}_{n_{\mathsf{PRG}} \in \mathbb{N}},$$
$$\left\{ (I, \mathbf{s}) \;\middle|\; \begin{array}{c} I \leftarrow \mathsf{GoodIdSamp}(1^{n_{\mathsf{PRG}}}), \\ \mathbf{s} \leftarrow U_{\ell(n_{\mathsf{PRG}})} \end{array} \right\}_{n_{\mathsf{PRG}} \in \mathbb{N}}.$$

- *We have* $\mu = \Omega(1)$ *and the following two distributions are identical:*
  - *Sample* $I \leftarrow \mathsf{IdSamp}(1^{n_{\mathsf{PRG}}})$.
  - *With probability* μ*, sample* $I \leftarrow \mathsf{GoodIdSamp}(1^{n_{\mathsf{PRG}}})$*. With probability* $1 - \mu$*, sample* $I \leftarrow \mathsf{BadIdSamp}(1^{n_{\mathsf{PRG}}})$.

Remark. Note that we do not require that it is possible to efficiently sample from GoodIdSamp or BadIdSamp; the distributions only need to exist. Looking ahead, this will be because we will ultimately instantiate polynomially many copies of our final FE scheme and then rely on an FE combiner [ABJ+19, JMS20]. In this case, we only require that the I in at least one of our copies was sampled from GoodIdSamp. Due to our reliance on SparseLPN, most of our security definitions in this paper will have this flavor.

3.4 LPN & Sparse LPN

Before defining our version of the sparse learning parity with noise (LPN) assumption, we first define the learning parity with noise (LPN) assumption, as used in [JLS22]. Let $\mathsf{Bern}(\mathbb{Z}_q, \eta)$ denote the distribution that is 0 with probability $1 - \eta$ and a uniformly random non-zero element of $\mathbb{Z}_q$ with probability η. For $q = 2$, this corresponds exactly to $\mathsf{Bern}(\eta)$.

Definition 6. *We say that (sub-exponential)* LPN over large fields *is true if the following holds. There exists a constant* $\delta \in (0,1)$ *such that for all constants* $\beta_1, \beta_2 > 0$ *such that* $q = q(n)$ *is a prime number of* n^{β_1} *bits and* $m = m(n) = n^{\beta_2}$*, the following two distributions are (sub-exponentially) indistinguishable:*

$$\left\{ (\mathbf{A}, \mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e}) \mid \mathbf{A} \leftarrow \mathbb{Z}_q^{m \times n}, \mathbf{s} \leftarrow \mathbb{Z}_q^n, \mathbf{e} \leftarrow \mathsf{Bern}(\mathbb{Z}_q, n^{-\delta})^m \right\}_{n \in \mathbb{N}},$$
$$\left\{ (\mathbf{A}, \mathbf{u}) \mid \mathbf{A} \leftarrow \mathbb{Z}_q^{m \times n}, \mathbf{u} \leftarrow \mathbb{Z}_q^m \right\}_{n \in \mathbb{N}}.$$

We define LPN only with respect to large fields because that is our only use for it (more specifically, for the construction of Preprocessed Polynomial Encodings (PPE) in [JLS22]).

We now introduce our $\mathsf{SparseLPN}$ assumption. Let $\mathsf{SparseMat}_q(t, n, m)$ denote the set of all matrices $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ such that every row of $\mathbf{A}$ has exactly t non-zero entries. For sparsity $t = O(1)$, since a uniformly random row-sparse matrix $\mathbf{A}$ is "bad" (i.e., has small dual distance) with noticeable probability $1/\mathsf{poly}(n)$, we only require the indistinguishability to hold with respect to some distribution of "good" sparse matrices $\mathsf{GoodSparseMat}_q(t, n, m)$ supported on (a subset) of $\mathsf{SparseMat}_q(t, n, m)$, as long as these good matrices are reasonably "dense" among the set of all sparse matrices.[5] Unfortunately, we do not know how to *efficiently* sample from such a $\mathsf{GoodSparseMat}_q(t, n, m)$ that has plausible sub-exponential security, as is needed to bootstrap to $i\mathcal{O}$ [BV18, AJ15, KNT22]. Instead, we will sample $\mathbf{A}$ from $\mathsf{SparseMat}_q(t, n, m)$ in our constructions and then use an unconditional FE combiner [ABJ$^+$19, JMS20] to get true sub-exponential security. We note that this issue also comes up in [JLS22] if instantiating the PRG in NC^0 with Goldreich's PRGs [Gol00] (see [JLS22, Remark 3.1]). More explicitly, we will assume that we can write the uniform distribution over $\mathsf{SparseMat}_q(t, n, m)$ as a mixture of (not necessarily efficiently sampleable) distributions $\mathsf{GoodSparseMat}_q(t, n, m)$ and $\mathsf{BadSparseMat}_q(t, n, m)$, with the weight on $\mathsf{GoodSparseMat}$ being $\Omega(1)$.

In general, our construction does not use any particular property of the uniform distribution over $\mathsf{SparseMat}_q(t, n, m)$, and as such, the distribution over $\mathsf{SparseMat}_q(t, n, m)$ can be replaced with any efficiently sampleable distribution $\mathsf{EffSparseMat}_q(t, n, m)$ over sparse matrices that satisfies the above assumption. For the case of sub-exponential security, we set $\mathsf{EffSparseMat}_q(t, n, m) = \mathsf{SparseMat}_q(t, n, m)$ only for concreteness. Additionally, even though $\mathsf{GoodSparseMat}_q(t, n, m)$ is not explicit, our constructions will be explicit because they use $\mathsf{EffSparseMat}_q(t, n, m)$. (That being said, one can specify a plausible "explicit" distribution for $\mathsf{GoodSparseMat}_q(t, n, m)$, but we do not know how to efficiently sample from it; see Appendix A.) However, for negligible (but *not* sub-exponential) security in the case of $\mathbb{Z}_2$, then there is an efficiently computable candidate distribution for $\mathsf{GoodSparseMat}_2(t, n, m)$ [AK23], which we denote by $\mathsf{AKSparseMat}_2(t, n, m)$. We define $\mathsf{SparseLPN}$ in a generic way using $\mathsf{EffSparseMat}$, to allow us to instantiate the assumption in different ways depending on whether we want negligible or sub-exponential security.

Definition 7. *We say that the (sub-exponential)* $\mathsf{SparseLPN}$ *assumption over* $\mathbb{Z}_q$ *is true if the following holds: there exist constants* $t \in \mathbb{N}$, $\delta \in (0, 1)$, *and* $\epsilon > 0$, *an efficiently sampleable distribution* $\mathsf{EffSparseMat}_q(t, n, m)$ *supported on a subset of* $\mathsf{SparseMat}_q(t, n, m)$, *and (not necessarily efficiently sampleable) distributions* $\mathsf{GoodSparseMat}_q(t, n, m), \mathsf{BadSparseMat}_q(t, n, m)$ *such that for* $m = m(n) = n^{1+\epsilon}$ *and* $\eta = \eta(n) = n^{-\delta}$, *the following distributions are (sub-exponentially) indistinguishable:*

$$\{(\mathbf{A}, \mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e} \pmod q) \mid \mathbf{A} \leftarrow \mathsf{GoodSparseMat}_q(t, n, m), \mathbf{s} \leftarrow \mathbb{Z}_q^n, \mathbf{e} \leftarrow \mathsf{Bern}(\mathbb{Z}_q, \eta)^m\}_{n \in \mathbb{N}},$$
$$\{(\mathbf{A}, \mathbf{u}) \mid \mathbf{A} \leftarrow \mathsf{GoodSparseMat}_q(t, n, m), \mathbf{u} \leftarrow \mathbb{Z}_q^m\}_{n \in \mathbb{N}},$$

as well as the mixture condition that

$$\mathsf{EffSparseMat}_q(t, n, m) = \mu \cdot \mathsf{GoodSparseMat}_q(t, n, m) + (1 - \mu) \cdot \mathsf{BadSparseMat}_q(t, n, m),$$

for some $\mu = \mu(n)$ *where* $\mu = \Omega(1)$. *Here, this equality refers to equality of distributions as a mixture distribution.*

[5]As a motivating example, for $q = 2$, the probability that two rows of $\mathbf{A}$ are the same is at least $1/\mathsf{poly}(n)$ if the sparsity t is constant. This immediately gives a vector with Hamming weight 2 in the left kernel of $\mathbf{A}$, breaking security.

If q is unspecified, we take it to mean $q = 2$, but more generally, we allow $q = q(n)$ to be a function of n.

For simplicity, without loss of generality, we will assume that the noise rate η is the inverse of a power of 2 (with exponent $O(\log n)$). To summarize, we have two distinct instantiations of the $\mathsf{SparseLPN}$ assumption depending on the security regime and modulus:

- If we only require negligible security and are restricting attention to $q = 2$, then we could plausibly take $\mu = 1$ and instantiate $\mathsf{EffSparseMat}_2(t, n, m) = \mathsf{AKSparseMat}_2(t, n, m)$.
- If we require sub-exponential security, then we can plausibly take $\mu = \Omega(1)$ and instantiate $\mathsf{EffSparseMat}_q(t, n, m) = \mathsf{SparseMat}_q(t, n, m)$.

  Note that for $t \geq 3$, choosing $\mathsf{GoodSparseMat}_q(t, n, m)$ to be matrices with sufficiently large dual distance, which has density $1 - 1/\mathsf{poly}(n)$ within $\mathsf{SparseMat}_q(t, n, m)$, the best known p.p.t. distinguishing attacks have sub-exponential advantage, as does any linear test. As such, we view setting $\mu = \Omega(1)$ as reasonably mild.

We refer to Appendix A for more detailed cryptanalysis on the assumption.

3.5 Linear Stretch PRG Family From Sparse LPN

In this section, we summarize a result from Applebaum, Ishai, and Kushilevitz [AIK08] that constructs a PRG family in NC^0 with linear stretch from the sparse LPN assumption (with different parameters).

Theorem 3.4 (Combining Construction 5.15, Lemma 5.16, Theorem 5.19 of [AIK08])**.** *Consider the following variant of $\mathsf{SparseLPN}$. For all $m = \Theta(n)$ and all constant noise rates $\eta = \Theta(1)$, suppose there exists $t = O(1)$ and a family of t-row sparse matrices $\mathbf{A} \in \mathbb{Z}_2^{m \times m}$ such that we have the computational indistinguishability*

$$\{(\mathbf{A}, \mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e} \pmod 2) \mid \mathbf{s} \leftarrow \mathbb{Z}_2^n, \mathbf{e} \leftarrow \mathsf{Bern}(\eta)^m\} \approx_c \{(\mathbf{A}, \mathbf{u}) \mid \mathbf{u} \leftarrow \mathbb{Z}_2^m\},$$

where the distributions cannot be distinguished by $\mathsf{poly}(n)$-time adversaries with advantage $\epsilon(n)$. Then, for some $\ell = \ell(n) = \Theta(n)$ there is an explicit family of NC^0 functions $G_{\ell,\mathbf{A}} : \{0,1\}^\ell \to \{0,1\}^{\ell+\Omega(\ell)}$ such that

$$\left\{(\mathbf{A}, G_{\ell,\mathbf{A}}(\mathbf{s})) \mid \mathbf{s} \leftarrow \{0,1\}^\ell\right\} \approx_c \left\{(\mathbf{A}, \mathbf{r}) \mid \mathbf{r} \leftarrow \{0,1\}^{\ell+\Omega(\ell)}\right\},$$

where the distributions cannot be distinguished by $\mathsf{poly}(n)$-time adversaries with advantage $\epsilon(n) + 2^{-\Omega(n)}$. (Here, by explicit, we mean that except for any advice needed to specify $\mathbf{A}$, the function $G_{\ell,\mathbf{A}}$ is in uniform NC^0.)

In particular, by composing this function with itself $O(1)$ times, this becomes a family of arbitrary linear-stretch PRGs in NC^0.

$\mathsf{SparseLPN}$ under this parameter regime is implied by our $\mathsf{SparseLPN}$ assumption (over $\mathbb{Z}_2$). This is because the PRG only needs a linear number of samples and constant noise rate, whereas our definition of $\mathsf{SparseLPN}$ implies hardness for a polynomial number of samples with inverse polynomial noise rate. (In the reduction, one can ignore extra samples and add appropriate noise to convert inverse polynomial noise into constant noise.) As a result, noting that the $\mathsf{SparseLPN}$ assumption and PRG definition (Definition 5) are compatible (in terms of $\mathsf{GoodIdSamp}/\mathsf{BadIdSamp}$ and $\mathsf{GoodSparseMat}/\mathsf{BadSparseMat}$), we have the following:

Corollary 3.5. *Suppose the (sub-exponential)* $\mathsf{SparseLPN}$ *assumption holds. Then, for any asymptotically linear stretch function $\ell(n) = \Theta(n)$, there is a (sub-exponentially) $\Omega(1)$-secure* PRG *family of stretch ℓ in* NC^0*, per Definition 5.*

4 Structured-Seed PRGs

The construction of $i\mathcal{O}$ by [JLS22] uses the existence of a sub-exponentially secure polynomial-stretch Boolean PRG in NC^0. Our central observation is that a weaker primitive suffices for the purposes of their construction, and that this primitive is achievable under the $\mathsf{SparseLPN}$ assumption. As it turns out, this primitive is closely related to the *structured-seed PRGs* (SPRG) defined and constructed by [JLS21]. We provide definitions below in a way that captures our construction, and we provide a comparison with the definition and construction of [JLS21] afterwards.

Definition 8. *A structured seed PRG (*SPRG *for short) consists of the following p.p.t. algorithms:*

- $I \leftarrow \mathsf{IdSamp}(1^{m_{\mathsf{SPRG}}})$*: the generation algorithm takes as input the output length m_{SPRG} in unary. It outputs a function index I.*
- $\mathsf{seed} \leftarrow \mathsf{SdSamp}(m_{\mathsf{SPRG}})$*: the preprocessing algorithm takes as input the desired output length m_{SPRG} in binary and outputs a binary string* seed*. The reason that m_{SPRG} is given in binary is that we will require the size of* SdSamp *to be $m_{\mathsf{SPRG}}^{1-\Omega(1)}$; note that we abuse notation and do* not *require this algorithm to run in polynomial time in* $\log(m_{\mathsf{SPRG}})$*.*
- $\mathsf{s} \leftarrow \mathsf{Eval}(I, \mathsf{seed})$*: deterministically outputs a string $\mathsf{s} \in \mathbb{Z}^{m_{\mathsf{SPRG}}}$.*

Intuitively, seed can serve as a compressed version of the SPRG output such that decompression can be described by a low-degree polynomial. In other words, we would like Eval to be such that computing $\mathsf{Eval}(I, \mathsf{SdSamp}(m_{\mathsf{SPRG}}))$ works just as well as computing $\mathsf{PRG}(U_n)$ for some $n = m_{\mathsf{SPRG}}^{1-\Omega(1)}$, for the purposes of the $i\mathcal{O}$ construction of [JLS22]. We next state the properties capturing this intuition:

Definition 9. *We say a* SPRG *satisfies (perfect) correctness if we have* $\mathsf{Eval}(I, \mathsf{seed}) \in \{0,1\}^{m_{\mathsf{SPRG}}}$ *for all I,* seed *in the support of* $\mathsf{IdSamp}, \mathsf{SdSamp}$ *respectively. (Note that this is not obvious, since the polynomial computing* Eval *is over $\mathbb{Z}$.)*

Definition 10. *We say* SPRG *is (sub-exponentially) $\mu(m_{\mathsf{SPRG}})$-secure if there exist distributions* $\mathsf{GoodIdSamp}(1^{m_{\mathsf{SPRG}}})$ *and* $\mathsf{BadIdSamp}(1^{m_{\mathsf{SPRG}}})$ *such that both of the following are true:*

- *The following distributions are (sub-exponentially) indistinguishable:*

$$\left\{ (I, \mathsf{Eval}(I, \mathsf{seed})) \;\middle|\; \begin{array}{c} I \leftarrow \mathsf{GoodIdSamp}(1^{m_{\mathsf{SPRG}}}), \\ \mathsf{seed} \leftarrow \mathsf{SdSamp}(m_{\mathsf{SPRG}}) \end{array} \right\}_{m_{\mathsf{SPRG}} \in \mathbb{N}}$$

$$\left\{ (I, \mathsf{s}) \;\middle|\; \begin{array}{c} I \leftarrow \mathsf{GoodIdSamp}(1^{m_{\mathsf{SPRG}}}), \\ \mathsf{s} \leftarrow U_{m_{\mathsf{SPRG}}} \end{array} \right\}_{m_{\mathsf{SPRG}} \in \mathbb{N}}$$

- *We have $\mu = \Omega(1)$ and the following two distributions are identical:*
  - *Sample* $I \leftarrow \mathsf{IdSamp}(1^{m_{\mathsf{SPRG}}})$.

– *With probability μ, sample $I \leftarrow \mathsf{GoodIdSamp}(1^{m_{\mathsf{SPRG}}})$. With probability $1-\mu$, sample $I \leftarrow \mathsf{BadIdSamp}(1^{m_{\mathsf{SPRG}}})$.*

Definition 11. *We say that an* SPRG *satisfies* ϵ-sublinear efficiency *if* SdSamp *is computable by a uniformly efficiently generatable randomized circuit of size at most $O(m_{\mathsf{SPRG}}^{1-\epsilon})$.*

Definition 12. *We say that a* SPRG *satisfies* degree d and τ-local decompression *if each entry of the output of* Eval *is expressible as a uniformly efficiently generatable polynomial over $\mathbb{Z}$ satisfying the following two requirements:*

- *Its total degree (over $\mathbb{Z}$) is at most d (which we will later require to be $O(1)$).*
- *It depends on at most $O(m_{\mathsf{SPRG}}^{\tau})$ entries of* seed.

We now point out some of the differences between our definition and the definition used in [JLS21]:

- The most significant difference is that we require SdSamp to satisfy sublinear efficiency. In contrast, [JLS21] is not concerned with the efficiency of SdSamp, but instead requires only that its output seed be small. (At the core, this is because the construction of [JLS21] is only aiming for sublinear *size*-succinct functional encryption; this is then bootstrapped to sublinear *time*-succinct FE assuming LWE [LPST16, GKP+13]. However our construction, like the later construction of [JLS22], does not assume LWE and hence needs to achieve sublinear time-succinctness directly.)
- The construction of [JLS21] allows SdSamp to generate a public and a private seed such that security still holds when the distinguisher is given the public seed. The reason for this is that they require Eval to be degree 2 in the private seed.

  In our case, we have some more freedom because it suffices for Eval to have degree $O(1)$, hence we do not need to work with a public seed.
- On the other hand, the fact that Eval has low locality is important for our use case, whereas locality is not important for [JLS21].
- [JLS21] assumes the polynomial computing Eval is over $\mathbb{Z}_p$ rather than $\mathbb{Z}$, where p is a prime determined by other components of their $i\mathcal{O}$ construction. This is a minor difference; we could also have defined SPRG to work over $\mathbb{Z}_p$ rather than $\mathbb{Z}$, but elected to work with $\mathbb{Z}$ to emphasize that our construction does not depend on the prime p at all.

It should also be noted that our SPRG and its application to $i\mathcal{O}$ is not analogous to that of [JLS21]; it is closer to the application of an NC^0 PRG in [JLS22]. To illustrate this, note firstly that the construction of [JLS21] of SPRG assumes both LPN over $\mathbb{Z}_p$ and the existence of a polynomial-stretch PRG in NC^0. This is the only place in the $i\mathcal{O}$ construction of [JLS21] where LPN over $\mathbb{Z}_p$ is used. In contrast, the construction of [JLS22] and our modification of it will use LPN over $\mathbb{Z}_p$ elsewhere, namely to hide the circuit C and PRG seed with a special-purpose homomorphic encryption scheme which will homomorphically carry out Yao's garbling procedure [Yao86].

Secondly, the notion of SPRG that we need for our purposes is actually *weaker* than a polynomial-stretch PRG in NC^0. Indeed, we can directly construct a simple SPRG given a sub-exponentially secure PRG G in NC^0 from n bits to m_{SPRG} bits, provided $m_{\mathsf{SPRG}} = n^{1+\Omega(1)}$:

- IdSamp deterministically outputs $I = \perp$.

- SdSamp simply outputs a uniform $\mathbf{r} \leftarrow U_n$.
- $\mathsf{Eval}(I, \mathbf{r})$ just directly evaluates $G(\mathbf{r})$.

SdSamp runs in size n which is sublinear in m_{SPRG}, and Eval has locality $O(1)$ which implies that it satisfies low-degree and τ-local decompression with $\tau = 0$. Moreover, this construction is clearly 1-secure.

We now formally state the theorem implied by our construction:

Theorem 4.1. *Assume the (sub-exponential)* $\mathsf{SparseLPN}$ *assumption holds (over $\mathbb{Z}_2$) for n and $m := m_{\mathsf{SPRG}} = n^{1+\epsilon}$. Then, there exists constants $\nu > 0$ and $d \in \mathbb{N}$ such that for any constant $\tau > 0$, there exists a perfectly correct (sub-exponentially) $\Omega(1)$-secure* SPRG *satisfying ν-sublinear efficiency with degree d and τ-local decompression.*

4.1 SPRG Construction Details

In essence, we will use $\mathsf{SparseLPN}$ as our SPRG and we will compress the sparse error vector $\mathbf{e}$ using low-rank decompositions of sparse matrices, as in [JLS21, JLS22]. We begin by setting up some parameters and notation:

- Let $\gamma = \delta/(1+\epsilon)$. Note that the noise rate $\eta = n^{-\delta} = m_{\mathsf{SPRG}}^{-\gamma}$. We also write $\eta = 2^{-b}$ for $b \in \mathbb{N}$ bounded by $O(\log m_{\mathsf{SPRG}})$.
- Set a constant parameter $\alpha \in (\gamma/2, 3\gamma/4)$ such that $2\alpha - \gamma < \tau$.
- Let $L = \lceil m_{\mathsf{SPRG}}^{\alpha} \rceil$ and $\ell = \lceil m_{\mathsf{SPRG}}^{1-2\alpha} \rceil$. Note that $2m_{\mathsf{SPRG}} \geq \ell \cdot L^2 \geq m_{\mathsf{SPRG}}$.
- Let $\beta \in (0, \alpha - \gamma/2)$ be a constant, and let $\rho = m_{\mathsf{SPRG}}^{\beta}$ be a slack parameter. Additionally, let $L' = \lfloor L^2 m_{\mathsf{SPRG}}^{-\gamma} + \rho \cdot L m_{\mathsf{SPRG}}^{-\gamma/2} \rfloor$. Note that $L' = O(m_{\mathsf{SPRG}}^{2\alpha-\gamma} + m_{\mathsf{SPRG}}^{\beta+\alpha-\gamma/2}) = O(m_{\mathsf{SPRG}}^{2\alpha-\gamma}) = O(m_{\mathsf{SPRG}}^{\tau})$.
- Let ϕ be a canonical injective map that maps $\ell \cdot L^2 \geq m_{\mathsf{SPRG}}$ items into ℓ matrices $\{\mathbf{M}_i\}_{i \in [\ell]}$ of size $L \times L$. That is, for $j \in [\ell \cdot L^2]$, $\phi(j) = (j_1, (j_2, j_3))$, where item j is mapped to matrix $j_1 \in [\ell]$ and element $(j_2, j_3) \in [L] \times [L]$ of the matrix. As noted by [JLS22], this map can be computed with a circuit of size $\mathsf{poly}(\log(\ell \cdot L^2)) = \mathsf{poly}(\log m_{\mathsf{SPRG}})$, by first dividing $j \in [\ell \cdot L^2]$ by ℓ and setting the remainder as j_1. Then the quotient can be further divided by L, yielding a quotient and remainder which can be used as (j_2, j_3).

  For convenience, we also use ψ to denote the second half of this injective map, namely the part that maps $j \in [L^2]$ to $(j_2, j_3) \in [L] \times [L]$.

Our construction is described in Figure 3.

SPRG Construction

$I \leftarrow \mathsf{IdSamp}(1^{m_{\mathsf{SPRG}}})$:

1. Sample $\mathbf{A} \leftarrow \mathsf{EffSparseMat}(t, n, m_{\mathsf{SPRG}})$.
2. Output $I = \mathbf{A}$.

$\mathsf{seed} \leftarrow \mathsf{SdSamp}(m_{\mathsf{SPRG}})$. We describe the procedure in pseudocode below, and defer a discussion of its implementation as a sublinear-time circuit to Section 4.2.

1. Sample errors $\mathbf{e} \leftarrow \mathsf{Bern}(\eta)^{\ell \cdot L^2}$. (We will actually sample from another distribution that is sub-exponentially close in statistical distance to $\mathsf{Bern}(\eta)^{\ell \cdot L^2}$, and we will also ensure that this distribution is supported on $\{0,1\}^{\ell \cdot L^2}$ for correctness.)
2. For every index $j \in [\ell]$, initialize a matrix $\mathbf{M}_j \in \{0,1\}^{L \times L} \subseteq \mathbb{Z}^{L \times L}$ with zero entries.
3. For each $j \in [\ell \cdot L^2]$, compute $\phi(j) = (j_1, (j_2, j_3))$ and set $\mathbf{M}_{j_1}[j_2, j_3] = \mathbf{e}_j$.
4. If there is any $j \in [\ell]$ such that the number of nonzero entries in $\mathbf{M}_j$ is outside the range $[L^2 m_{\mathsf{SPRG}}^{-\gamma} - \rho \cdot L m_{\mathsf{SPRG}}^{-\gamma/2}, L^2 m_{\mathsf{SPRG}}^{-\gamma} + \rho \cdot L m_{\mathsf{SPRG}}^{-\gamma/2}]$, return to Step 1 and resample $\mathbf{e}$. (By the way that we are implementing the sampling of $\mathbf{e}$, we will actually ensure that this condition will never be violated, so $\mathbf{e}$ will never have to be resampled.)
5. For each $j \in [\ell]$, compute matrices $\mathbf{U}_j, \mathbf{V}_j^\top \in \{0,1\}^{L \times L'} \subseteq \mathbb{Z}^{L \times L'}$ such that $\mathbf{M}_j = \mathbf{U}_j \cdot \mathbf{V}_j$ (where this matrix multiplication takes place over $\mathbb{Z}$).
6. Finally, sample $\mathbf{r} \leftarrow \{0,1\}^n$ and output $\mathsf{seed} = (\mathbf{r}, \{\mathbf{U}_j, \mathbf{V}_j\}_{j \in [\ell]})$.

$\mathbf{s} \leftarrow \mathsf{Eval}(I, \mathsf{seed})$:

1. Parse $I = \mathbf{A}$ and $\mathsf{seed} = (\mathbf{r}, \{\mathbf{U}_j, \mathbf{V}_j\}_{j \in [\ell]})$.
2. For each $j \in [\ell]$, compute $\mathbf{M}_j = \mathbf{U}_j \cdot \mathbf{V}_j$.
3. Define $\mathbf{e} \in \{0,1\}^{m_{\mathsf{SPRG}}}$ as follows: for each $j \in [m_{\mathsf{SPRG}}]$, compute $\phi(j) = (j_1, (j_2, j_3))$ and set $\mathbf{e}_j = \mathbf{M}_{j_1}[j_2, j_3]$.
4. Output $\mathbf{s} = \mathbf{A}\mathbf{r} + \mathbf{e} \bmod 2$.

Figure 3: Our SPRG construction.

Security: The main observation is the following lemma:

Lemma 4.2. *The probability that the number of nonzero entries in any $\mathbf{M}_j$ is outside the range $[L^2 m_{\mathsf{SPRG}}^{-\gamma} - \rho \cdot L m_{\mathsf{SPRG}}^{-\gamma/2}, L^2 m_{\mathsf{SPRG}}^{-\gamma} + \rho \cdot L m_{\mathsf{SPRG}}^{-\gamma/2}]$ is $O(\exp(-m_{\mathsf{SPRG}}^{\Omega(1)}))$.*

Proof. It suffices to show the result for a fixed $j \in [\ell]$; the conclusion would then follow from a union bound since $\ell = m_{\mathsf{SPRG}}^{O(1)}$. Following Claim 4.2 in [JLS22], this follows from a straightforward Chernoff bound. For each $j_2, j_3 \in [L]$, let X_{j_2,j_3} be 1 if $\mathbf{M}_j[j_2, j_3] = 1$ and 0 otherwise. Then the X_{j_2,j_3}'s are i.i.d. samples from $\mathsf{Bern}(m_{\mathsf{SPRG}}^{-\gamma})$. Hence their sum has expectation $L^2 m_{\mathsf{SPRG}}^{-\gamma}$. A Chernoff bound then tells us that:

$$\Pr\left[\left|\sum_{j_2,j_3\in[L]} X_{j_2,j_3} - L^2 m_{\mathsf{SPRG}}^{-\gamma}\right| \geq \frac{\rho}{L m_{\mathsf{SPRG}}^{-\gamma/2}} \cdot L^2 m_{\mathsf{SPRG}}^{-\gamma}\right] \leq 2\exp\left(-\left(\frac{\rho}{L m_{\mathsf{SPRG}}^{-\gamma/2}}\right)^2 \cdot \frac{L^2 m_{\mathsf{SPRG}}^{-\gamma}}{3}\right)$$
$$= 2\exp\left(-\frac{\rho^2}{3}\right),$$

which implies the conclusion since $\rho = m_{\mathsf{SPRG}}^{\beta}$. Note the Chernoff bound applies in this setting because $\frac{\rho}{L m_{\mathsf{SPRG}}^{-\gamma/2}} = O(m_{\mathsf{SPRG}}^{\beta-\alpha+\gamma/2}) \in (0,1)$. □

It follows that that the $\mathbf{e}$ used when constructing the $\mathbf{U}_j, \mathbf{V}_j$ matrices is statistically close to an honest sample from $\mathsf{Bern}(\eta)^{m_{\mathsf{SPRG}}}$ with statistical distance bounded by $O(\exp(-m_{\mathsf{SPRG}}^{\Omega(1)}))$. Security is now immediate from the $\mathsf{SparseLPN}$ assumption; we will take $\mathsf{GoodIdSamp}$ to sample $\mathbf{A} \leftarrow \mathsf{GoodSparseMat}(t, n, m_{\mathsf{SPRG}})$ and $\mathsf{BadIdSamp}$ to sample $\mathbf{A} \leftarrow \mathsf{BadSparseMat}(t, n, m_{\mathsf{SPRG}})$.

Low-degree and τ-local decompression: We begin by observing that for any positive integer k, any function $F : \{0,1\}^k \to \{0,1\}$ can be computed by a multilinear polynomial over $\mathbb{Z}$ (which hence has total degree at most k). We now argue step by step:

- The ith entry of $\mathbf{Ar}$ mod 2 is an inner product of a row $\mathbf{a}_i$ of $\mathbf{A}$ with $\mathbf{r}$ reduced modulo 2. Because $\mathbf{a}_i$ has $t = O(1)$ nonzero entries, it follows that this is a function of t entries of $\mathbf{r}$. By our observation, this is expressible as a polynomial over $\mathbb{Z}$ evaluated on $\mathbf{r}$ with locality and total degree at most t.

- For each $j \in [m_{\mathsf{SPRG}}]$, the jth entry of $\mathbf{e}$ is $\mathbf{M}_{j_1}[j_2, j_3]$ where $\phi(j) = (j_1, (j_2, j_3))$. This is in turn equal to $(\mathbf{U}_{j_1} \cdot \mathbf{V}_{j_1})[j_2, j_3] = \sum_{i\in[L']} \mathbf{U}_{j_1}[j_2, i] \cdot \mathbf{V}_{j_1}[i, j_3]$. As a function of all the entries of the matrices $\mathbf{U}_{j_1}$ and $\mathbf{V}_{j_1}$, this has degree 2 and locality $O(L') = O(m_{\mathsf{SPRG}}^{\tau})$.

- Each entry of $\mathbf{Ar} + \mathbf{e}$ mod 2 is the XOR of one entry from each of $\mathbf{Ar}$ mod 2 and $\mathbf{e}$. By the aforementioned observation, this is expressible as a polynomial over $\mathbb{Z}$ evaluated on ($\mathbf{Ar}$ mod 2) and $\mathbf{e}$ with locality and total degree ≤ 2.

Putting these together and using Lemmas 3.1 and 3.2 implies that each entry of $\mathsf{Eval}(I, \mathsf{seed})$ is expressible as a polynomial over $\mathbb{Z}$ in the entries of seed with total degree at most $4t = O(1)$ and locality $O(m_{\mathsf{SPRG}}^{\tau})$. Note importantly that our bound $4t$ on the degree comes directly from the parameters in the $\mathsf{SparseLPN}$ assumption and is hence independent of τ.

4.2 Sublinear-Time Seed Sampling

It remains to argue the sublinear efficiency of SdSamp. We begin with some lemmas about circuit implementability that will be helpful throughout this section.

4.2.1 Circuit Implementability Lemmas

We begin by recalling a result about circuits for sorting [AKS83]:

Lemma 4.3 ([AKS83], as stated in Lemma 4.4 of [JLS22])**.** *Suppose we have N strings of size B bits and a comparator circuit $\psi : \{0,1\}^B \times \{0,1\}^B \to \{0,1\}$ of size T_ψ. Then these strings can be sorted with respect to the comparison function computed by ψ with a uniformly efficiently generatable circuit of size $O(N \cdot B \cdot T_\psi \cdot \mathsf{poly}(\log(N \cdot B \cdot T_\psi)))$.*

We also recall the following lemma that will enable us to work with constant-tape Turing machines instead of circuits:

Lemma 4.4 ([PF79], as stated in Lemma 4.5 of [JLS22])**.** *For any Turing machine M with $O(1)$ tapes running in time $T(n)$ on inputs of length n, there exists an efficiently generatable Boolean circuit family $\{\mathcal{C}_n\}_{n\in\mathbb{N}}$ where $\mathcal{C}_n$ takes n bits of input, produces the same output, and has $O(T(n) \cdot \mathsf{poly}(\log T(n)))$ gates.*

We now present some simple sampling lemmas that we will use in Section 4.2.2 to implicitly sample the error vector $\mathbf{e}$ in sublinear size.

Lemma 4.5. *Let N be a positive integer. Let $\mathcal{D}$ be any distribution supported on $\{0, 1, \ldots, N-1\}$. Then for any $\epsilon > 0$, there exists a (randomized) circuit of size $O(N^{1+\epsilon})$ that generates a sample from $\mathcal{D}$ with statistical error $O(\exp(-N^{\Omega(1)}))$. (In other words, the distribution of the circuit's output is supported on $\{0, \ldots, N-1\}$ and has statistical distance $O(\exp(-N^{\Omega(1)}))$ from $\mathcal{D}$.)*

Moreover, if $\mathcal{D}$ is the uniform distribution on $\{0, 1, \ldots, N-1\}$, there exists such a circuit of size only $O(N^\epsilon)$.

Additionally, if $p_i := \Pr_{X\sim\mathcal{D}}[X = i]$ is uniformly and efficiently computable (represented as a rational number) for all $i \in [0, N-1]$, then this circuit is uniformly and efficiently generatable.

Proof. Let $\delta > 0$ be a parameter we set at the end, and define $M = \lceil N^\delta \rceil$. For each $i \in [0, N-1]$, let $p_i = \Pr_{X\sim\mathcal{D}}[X = i]$. Additionally, define $q_i = 2^{-M} \cdot \lfloor 2^M p_i \rfloor$ to be the largest multiple of 2^{-M} that is at most p_i. We hence have $\sum_i q_i \le 1$. The only source of non-uniformity in our circuit construction will be the use of the q_i's, which are efficiently computable from the p_i's. This establishes the final claim in the statement of the lemma about uniform generatability. We proceed as follows:

1. Sample a uniformly random integer x from $\{0, 1, \ldots, 2^M - 1\}$.
2. Output the smallest integer $i \in [0, N-1]$ such that $2^M \cdot \sum_{j\le i} q_j > x$. If no such i exists, output 0.

We first argue correctness. For any $i > 0$, the probability that we sample i is:

$$\begin{aligned}\Pr\left[\left(2^M \cdot \sum_{j\le i} q_j > x\right) \wedge \left(2^M \cdot \sum_{j\le i-1} q_j \le x\right)\right] &= \frac{1}{2^M} \cdot \left(2^M \cdot \sum_{j\le i} q_j - 2^M \cdot \sum_{j\le i-1} q_j\right) \\ &= q_i.\end{aligned}$$

Hence the probability that we sample $i = 0$ is $q_0' := 1 - \sum_{i=1}^{N-1} q_i$.

We find that the total variation distance between these two distributions is:

$$\begin{aligned}
\frac{1}{2}\left(|p_0 - q_0'| + \sum_{i=1}^{N-1}|p_i - q_i|\right) &= \frac{1}{2}\sum_{i=1}^{N-1}(p_i - q_i) + \frac{1}{2}\left|1 - \sum_{i=1}^{N-1} q_i - p_0\right| \\
&\leq \frac{1}{2}\sum_{i=1}^{N-1}(p_i - q_i) + \frac{1}{2}\left(\left|1 - \sum_{i=0}^{N-1} q_i\right| + |p_0 - q_0|\right) \\
&= \frac{1}{2}\sum_{i=0}^{N-1}(p_i - q_i) + \frac{1}{2}\left(1 - \sum_{i=0}^{N-1} q_i\right) \\
&= \sum_{i=0}^{N-1}(p_i - q_i) \\
&\leq N \cdot 2^{-M} \\
&= O(\exp(-N^{\Omega(1)})),
\end{aligned}$$

thus establishing correctness.

It remains to argue the efficiency of our algorithm when implemented as a circuit. Step 1 can directly be done in size $O(M) = O(N^\delta)$. To address the efficiency of Step 2, we need to handle the case where $\mathcal{D}$ is the uniform distribution differently, so we split into two cases:

- If $\mathcal{D}$ is the uniform distribution, then we have $p_i = 1/N \Rightarrow q_i = 2^{-M} \cdot \lfloor 2^M/N \rfloor$ for all i. Call this constant q. Then the output of the circuit is the smallest integer $i \in [0, N]$ such that $2^M \cdot \sum_{j\leq i} q_j > x \Leftrightarrow 2^M \cdot (i+1)q > x \Leftrightarrow (i+1) \cdot \lfloor 2^M/N \rfloor > x$. This i can be found directly with a division circuit of size $\mathsf{poly}(M) = \mathsf{poly}(N^\delta)$, where poly here denotes a fixed polynomial.

  The total size of our sampling circuit is hence $O(N^\delta + \mathsf{poly}(N^\delta))$, which is $O(N^\epsilon)$ for δ a sufficiently small constant.

- If $\mathcal{D}$ is not necessarily uniform, then we proceed as follows. We would like to show that there exists an efficient circuit that, given x and the integers $2^M q_0, \ldots, 2^M q_N$ as inputs, outputs i. To do this, we argue by considering a three-tape Turing machine.

  The first tape will store the sequence $2^M q_0, \ldots, 2^M q_N$, the second tape will initially store x, and the third tape will store a counter i which is initialized to 0. We will repeat the following while $i \leq N$: subtract $2^M q_i$ from the value on the second tape. If the result is negative, we will output i and terminate. Otherwise, we will increment i by 1 and continue.

  The loop runs for at most $N+1$ steps and each step will run in time $\mathsf{poly}(M)$ (due to arithmetic operations on M-bit integers). Note also that the head on the first tape only makes a single forward pass. Hence the runtime of this TM is $O(N \cdot \mathsf{poly}(M)) = O(N \cdot \mathsf{poly}(N^\delta))$.

  It follows by Lemma 4.4 that there exists a circuit of size $O(N \cdot \mathsf{poly}(N^\delta) \cdot \mathsf{poly}(\log N))$ that realizes the same functionality for Step 2.

  The total size of our sampling circuit is hence $O(N^\delta + N \cdot \mathsf{poly}(N^\delta) \cdot \mathsf{poly}(\log N))$, which is $O(N^{1+\epsilon})$ for δ a sufficiently small constant.

□

Finally, we show that polynomially sparse Hamming slices can be sampled with a sublinear-size circuit. To do this, we begin with the following straightforward lemma:

Lemma 4.6. *Let $0 < \epsilon < \delta < 1$ be constants and N a positive integer, and let $k = \lceil N^{1-\epsilon} \rceil$. Suppose we sample $x_1, \ldots, x_k$ independently and uniformly from $\{0, 1, \ldots, N-1\}$ with replacement. Then for any constant $c \geq 1$, the number of distinct elements in $\{x_1, \ldots, x_k\}$ is $\geq c \cdot N^{1-\delta}$, with probability $1 - O(\exp(-N^{\Omega(1)}))$.*

Proof. If the number of distinct elements in $\{x_1, \ldots, x_k\}$ is $< cN^{1-\delta}$, then there must exist a set $S \subseteq \{0, 1, \ldots, N-1\}$ of size $\lfloor cN^{1-\delta} \rfloor$ such that $x_i \in S$ for all $i \in [k]$. We will take a union bound over all such sets S.

Firstly for a fixed set S, the probability that $x_i \in S$ for all $i \in [k]$ is:

$$\begin{aligned} \left(\frac{|S|}{N}\right)^k &\leq c^k N^{-\delta k} \\ &= \exp(-\delta k \log N + k \log c) \\ &\leq \exp(-\delta N^{1-\epsilon}). \end{aligned}$$

Secondly the number of such sets S is:

$$\begin{aligned} \binom{N}{\lfloor cN^{1-\delta} \rfloor} &\leq N^{\lfloor cN^{1-\delta} \rfloor} \\ &\leq \exp(cN^{1-\delta} \log N). \end{aligned}$$

A union bound then tells us that the probability that the x_i's do not cover $N^{1-\delta}$ distinct elements is at most:

$$\begin{aligned} \exp(cN^{1-\delta} \log N) \cdot \exp(-\delta N^{1-\epsilon}) &= \exp(cN^{1-\delta} \log N - \delta N^{1-\epsilon}) \\ &\leq \exp(-N^{1-\delta}) \end{aligned}$$

for N sufficiently large, as desired. □

We can now describe our circuit for sampling sparse Hamming slices:

Lemma 4.7. *Let $\delta \in (0, 1)$ be a constant and N, k be positive integers such that $k \leq O(N^{1-\delta})$. Then there exists a (randomized) circuit of size $O(N^{1-\delta/4})$ that takes k as input and outputs a sample from a distribution over subsets of $\{0, 1, \ldots, N-1\}$ of size at most k, that has statistical distance $O(\exp(-N^{\Omega(1)}))$ from the uniform distribution over all subsets of size exactly k.*

The set will be output as a list, padded with $\perp$ to some length $N'' = O(N^{1-\delta})$. (We do not require that the list be sorted, or that the order of the elements in the list is distributed in any specific way.)

Proof. Let $\epsilon \in (3\delta/4, \delta)$ be a constant, and let $N' = \lceil N^{1-\epsilon} \rceil$ and $M = \lceil N^{\epsilon/3} \rceil$. The idea is to use Lemma 4.6 to sample many elements of $\{0, 1, \ldots, N-1\}$ with replacement, and then filter out duplicates. Our circuit proceeds as follows:

1. Use Lemma 4.5 repeatedly to obtain N' independent uniform samples $x_1, \ldots, x_{N'}$ from $\{0, 1, \ldots, N-1\}$.
2. Using Lemma 4.3, sort these samples in ascending order. We now have a list $x_{\sigma(1)} \leq \ldots \leq x_{\sigma(N')}$, for some permutation σ.

3. Construct the filtered list $\left\{x_{\sigma(j)} : j = 1 \text{ or } x_{\sigma(j)} \neq x_{\sigma(j-1)}\right\}$ and let its length be $c \leq N'$. These entries will all be distinct. We now have a list $x_{\sigma(j_1)} < \ldots < x_{\sigma(j_c)}$. Pad this list on the right with $\perp$ so that it has length exactly N'.

4. Sample uniformly random integers $r_1, r_2, \ldots, r_c \in \left\{0, 1, \ldots, 2^M - 1\right\}$. (Once again, pad this on the right with $\perp$ so that it has length exactly N'.)

5. Construct the tuples $(r_1, x_{\sigma(j_1)}), \ldots, (r_c, x_{\sigma(j_c)})$ (padded with $\perp$ to length N').

6. Using Lemma 4.3, sort these tuples in ascending order of first entry (ties can be broken arbitrarily), keeping $\perp$ symbols at the end. We now have the tuples $(r_{\sigma'(1)}, x_{\sigma(j_{\sigma'(1)})}), \ldots, (r_{\sigma'(c)}, x_{\sigma(j_{\sigma'(c)})})$ for some permutation σ' on $[c]$. (Once again, this will be padded with $\perp$ to length N'.)

7. Construct $\left\{x_{\sigma(j_{\sigma'(1)})}, \ldots, x_{\sigma(j_{\sigma'(c)})}, \perp, \ldots, \perp\right\}$ (this is padded to length N').

8. Replace all entries after the kth symbol in this list with $\perp$, and truncate after $N'' \leq N'$ symbols.

We first address correctness.

- In Step 1, we obtain N' independent uniform samples with statistical error at most $N' \cdot O(\exp(-N^{\Omega(1)})) = O(\exp(-N^{\Omega(1)}))$.

- After Steps 2 and 3, we will have a sorted list $y_1 < \ldots < y_c$ of all distinct values that were sampled. Conditioned on the value of c, it is clear that $\{y_1, \ldots, y_c\}$ is a uniformly random subset of $\{0, 1, \ldots, N-1\}$ of size c.

- In Steps 4, 5, and 6, we use the random integers r_i to randomly shuffle $y_1, \ldots, y_c$, so that we now have a sequence $z_1, \ldots, z_c$ that is a uniformly random sequence of c distinct elements of $\{0, 1, \ldots, N-1\}$.

  Note that this shuffling argument relies on the r_i's being distinct; by a union bound, the probability that this is not the case is $\leq \binom{c}{2} \cdot 2^{-M} \leq N^2 \cdot 2^{-N^{\epsilon/3}} = O(\exp(-N^{\Omega(1)}))$, which is within our tolerance for statistical error.

- Since $\epsilon < \delta$, Lemma 4.6 tells us that we will have $c \geq O(N^{1-\delta}) \geq k$ with probability $1 - \exp(-N^{\Omega(1)})$. It follows that we will output $(z_1, \ldots, z_k, \perp, \ldots, \perp)$ with statistical error $\exp(-N^{\Omega(1)})$, which by the above is a uniformly random subset of $\{0, 1, \ldots, N-1\}$ of size k, as desired.

  Even if $c < k$, our output will just be $(z_1, \ldots, z_c, \perp, \ldots, \perp)$, which corresponds to some subset of size at most k from $\{0, 1, \ldots, N-1\}$.

Next, we address circuit implementability:

- Using the parameter $\epsilon/2$ in Lemma 4.5, it follows that Step 1 can be carried out with a circuit of size $N' \cdot O(N^{\epsilon/2}) = O(N^{1-\epsilon/2})$.

- In Step 2, we are sorting N' strings of size $O(\log N)$ with a comparator circuit of size $O(\log N)$. By Lemma 4.3, this can be achieved with a circuit of size $O(N' \cdot \mathsf{poly}(\log N)) = O(N^{1-\epsilon} \cdot \mathsf{poly}(\log N))$.

- Step 3 can be implemented using a three-tape Turing machine. We can load the input $x_{\sigma(1)}, \ldots, x_{\sigma(N')}$ on both the first and second tape and then make a single pass over the two tapes in parallel (while the head on the first tape is at $x_{\sigma(i)}$, the head on the second would be at $x_{\sigma(i+1)}$). During this time, the output can be written to a third tape. This Turing machine runs in time $O(N' \cdot \mathsf{poly}(\log N))$, so it follows by Lemma 4.4 that this can be implemented with a circuit of size $O(N^{1-\epsilon} \cdot \mathsf{poly}(\log N))$.

- The sampling in Step 4 can be done directly since the number of elements being sampled from is a power of 2; the size of this circuit is $O(N' \cdot M) = O(N^{1-2\epsilon/3})$.

- Step 5 can be implemented using a three-tape Turing machine. We can load the inputs $r_1, \ldots, r_{N'}$ on the first tape and $x_{\sigma(j_1)}, \ldots, x_{\sigma(j_c)}$ (padded to length N') on the second tape, and then make a single pass over the two tapes in parallel while writing the output to a third tape. This Turing machine runs in time $O(N' \cdot M) = O(N^{1-2\epsilon/3})$. By Lemma 4.4, it follows that this can be implemented in a circuit of size $O(N^{1-2\epsilon/3} \cdot \mathsf{poly}(\log N))$.

- In Step 6, we are sorting N' strings of size $O(M)$ using a comparator circuit of size $O(M)$. By Lemma 4.3, this can be done with a circuit of size $O(N' \cdot M^2 \cdot \mathsf{poly}(\log N)) = O(N^{1-\epsilon/3} \cdot \mathsf{poly}(\log N))$.

- Step 7 is trivially implementable with a two-tape Turing machine, making a single pass over the input. This Turing machine runs in time $O(N' \cdot M) = O(N^{1-2\epsilon/3})$. Hence Lemma 4.4 tells us that this can be implemented with a circuit of size $O(N^{1-2\epsilon/3} \cdot \mathsf{poly}(\log N))$.

- Step 8 is easily implementable with a three-tape Turing machine. One input tape will contain the inputs k and N'' and the second input tape will contain the output of Step 7. The output will be written to the third tape. This Turing machine runs in time $O(N' \cdot \mathsf{poly}(\log N))$, so it follows by Lemma 4.4 that this can be implemented with a circuit of size $O(N^{1-\epsilon/3} \cdot \mathsf{poly}(\log N))$.

The overall size of our circuit is hence $O(N^{1-\epsilon/3} \cdot \mathsf{poly}(\log N)) \leq O(N^{1-\delta/4})$, as desired. $\square$

4.2.2 Implementation of SdSamp

We need to implement SdSamp using a circuit with size sublinear in m_{SPRG}, so we cannot afford to sample the Bernoulli error vector directly. Instead, we will first sample its Hamming weight for each of the ℓ buckets, and then sample the set of positions where it is 0.

To this end, let $\mathcal{D}$ be the distribution of $x \leftarrow \mathsf{Binom}(L^2, \eta) = \mathsf{Binom}(L^2, m_{\mathsf{SPRG}}^{-\gamma})$ conditioned on $x \in [L^2 m_{\mathsf{SPRG}}^{-\gamma} - \rho \cdot L m_{\mathsf{SPRG}}^{-\gamma/2}, L^2 m_{\mathsf{SPRG}}^{-\gamma} + \rho \cdot L m_{\mathsf{SPRG}}^{-\gamma/2}]$. By Lemma 4.2, this is $O(\exp(-m_{\mathsf{SPRG}}^{\Omega(1)}))$-close in statistical distance to $\mathsf{Binom}(L^2, m_{\mathsf{SPRG}}^{-\gamma})$. Moreover, let $\mathcal{D}'$ be the translated distribution $\mathcal{D} - \lceil L^2 m_{\mathsf{SPRG}}^{-\gamma} - \rho \cdot L m_{\mathsf{SPRG}}^{-\gamma/2} \rceil$. This is just for compatibility with Lemma 4.5.

For each $j \in [\ell]$, our goal is to independently sample matrices $\mathbf{U}_j, \mathbf{V}_j \in \mathbb{Z}^{L \times L'}$ such that the entries of $\mathbf{U}_j \cdot \mathbf{V}_j$ are independent samples from $\mathsf{Bern}(\eta)$ (up to sub-exponential statistical error). We do this as follows:

1. Using Lemma 4.5, sample $k' \leftarrow \mathcal{D}'$.

2. Compute $k = k' + \lceil L^2 m_{\mathsf{SPRG}}^{-\gamma} - \rho \cdot L m_{\mathsf{SPRG}}^{-\gamma/2} \rceil$. Note by definition of $\mathcal{D}$ that we must have $k \leq L' = O(m_{\mathsf{SPRG}}^{2\alpha-\gamma}) = O(L^{2(1-\gamma/2\alpha)})$.

3. Hence using Lemma 4.7, we can sample a subset $S \subseteq \{0, 1, \ldots, L^2 - 1\}$ of size $c \leq k$, padded on the right to length $L' \geq k$ with $\perp$ symbols.

   With all but sub-exponential probability, we will have $c = k$ and S will be a uniform sample from subsets of $\{0, 1, \ldots, L^2 - 1\}$ of size exactly k. Label the elements of S as $(s_1, s_2, \ldots, s_c)$.

4. For each $i \in [c]$, compute $(x_i, y_i) = \psi(s_i)$, so that $x_i, y_i \in [L]$. Let $\mathbf{u}_i \in \mathbb{Z}^L$ be 1 in position x_i and 0 everywhere else, and similarly let $\mathbf{v}_i \in \mathbb{Z}^L$ be 1 in position y_i and 0 everywhere else. Once again, we pad these arrays on the right to length L' with $\perp$ symbols.

5. Define $\mathbf{U}_j = \begin{bmatrix}\mathbf{u}_1 & \ldots & \mathbf{u}_c & \mathbf{0}^{L\times(L'-c)}\end{bmatrix}$ and $\mathbf{V}_j^\top = \begin{bmatrix}\mathbf{v}_1 & \ldots & \mathbf{v}_c & \mathbf{0}^{L\times(L'-c)}\end{bmatrix}$.

Once we have done this for each $j \in [\ell]$, we can just sample $\mathbf{r} \leftarrow U_n$ directly and output $(\mathbf{r}, \{\mathbf{U}_j, \mathbf{V}_j\}_{j\in[\ell]})$.

Correctness: First, we address the statistical errors incurred from calls to Lemmas 4.5 and 4.7. In the calls to Lemma 4.5, we are taking $N = L' - (L^2 m_{\mathsf{SPRG}}^{-\gamma} - \rho \cdot L m_{\mathsf{SPRG}}^{-\gamma/2}) = \Theta(\rho \cdot L m_{\mathsf{SPRG}}^{-\gamma/2}) = \Theta(m_{\mathsf{SPRG}}^{\beta+\alpha-\gamma/2}) = \Theta(m_{\mathsf{SPRG}}^{\Omega(1)})$. Hence the statistical error is $O(\exp(-m_{\mathsf{SPRG}}^{\Omega(1)}))$. In the calls to Lemma 4.7, we are taking $N = L^2 = \Theta(m_{\mathsf{SPRG}}^{2\alpha})$, so here also we incur statistical error is $O(\exp(-m_{\mathsf{SPRG}}^{\Omega(1)}))$. We make at most $\ell < m_{\mathsf{SPRG}}$ calls to each of these lemmas, so the total statistical error will remain sub-exponentially small.

Up to the statistical error mentioned above, we have sampled the Hamming weight $k \leftarrow \mathsf{Binom}(L^2, m_{\mathsf{SPRG}}^{-\gamma})$ and then sampled a uniformly random subset of $\{0, 1, \ldots, L^2 - 1\}$ of size k. This is clearly equivalent to including each element of $\{0, 1, \ldots, L^2 - 1\}$ in S independently with probability η each.

Finally, we argue that $\mathbf{U}_j \cdot \mathbf{V}_j$ is 1 in row x and column y if $\psi^{-1}(x, y) \in S$ and 0 otherwise. This will complete our argument. Indeed, we have $\mathbf{U}_j \cdot \mathbf{V}_j = \sum_{i=1}^{k} \mathbf{u}_i \mathbf{v}_i^\top$. The matrix $\mathbf{u}_i \mathbf{v}_i^\top$ is 1 in row x_i and column y_i and 0 everywhere else. Since $s_1, \ldots, s_c$ are distinct and hence the tuples (x_i, y_i) are distinct, our claim follows.

Even in the case where we encounter some statistical error when calling Lemmas 4.5 and 4.7, we will always sample a collection $(s_1, \ldots, s_c)$ of $c \leq L'$ distinct elements of $\{0, 1, \ldots, L^2 - 1\}$, for some c. It then follows that all entries of $\mathbf{U}_j \cdot \mathbf{V}_j$ will still be 0 or 1, which ensures perfect correctness of our SPRG construction.

Uniform Generatability: The only thing that needs to be addressed is the call to Lemma 4.5, for which we need to check that the histogram of $\mathcal{D}'$ can be efficiently computed. To do this, it clearly suffices to efficiently compute the histogram of $\mathsf{Binom}(L^2, m_{\mathsf{SPRG}}^{-\gamma})$; we can then apply the necessary conditioning to recover a histogram for $\mathcal{D}$, and finally translate it to obtain a histogram for $\mathcal{D}'$. This histogram is characterized by the following identity:

$$\begin{aligned}
\Pr\left[X = i \mid X \leftarrow \mathsf{Binom}(L^2, m_{\mathsf{SPRG}}^{-\gamma})\right] &= \binom{L^2}{i} \cdot \left(m_{\mathsf{SPRG}}^{-\gamma}\right)^i \cdot \left(1 - m_{\mathsf{SPRG}}^{-\gamma}\right)^{L^2-i} \\
&= \binom{L^2}{i} \cdot \left(2^{-b}\right)^i \cdot \left(1 - 2^{-b}\right)^{L^2-i} \\
&= \binom{L^2}{i} \cdot 2^{-bL^2} \cdot \left(2^b - 1\right)^{L^2-i}.
\end{aligned}$$

It is now clear that this is computable in time $\mathsf{poly}(m_{\mathsf{SPRG}})$; the binomial coefficients $\binom{L^2}{i}$ can all be computed in time $\mathsf{poly}(L) = \mathsf{poly}(m_{\mathsf{SPRG}})$ e.g., by iterating through Pascal's triangle. Since

$b \leq O(\log m_{\mathsf{SPRG}})$, we have $2^b - 1 \leq \mathsf{poly}(m_{\mathsf{SPRG}})$. The value $\left(2^b - 1\right)^{L^2-i}$ can hence be computed exactly, e.g., via repeated squaring; note importantly that the bit length of this number will always be at most $L^2 \cdot O(\log m_{\mathsf{SPRG}}) = \mathsf{poly}(m_{\mathsf{SPRG}})$. Finally, the factor of 2^{-bL^2} can be easily handled by bit shifting, noting again that $bL^2 = \mathsf{poly}(m_{\mathsf{SPRG}})$.

Sublinear Efficiency: We first argue that for any fixed $j \in [\ell]$, $\mathbf{U}_j$ and $\mathbf{V}_j$ can be sampled with a randomized circuit of size $O(m_{\mathsf{SPRG}}^{2\alpha-\Omega(1)})$:

- The initial call to Lemma 4.5 sets $N = \Theta(m_{\mathsf{SPRG}}^{\beta+\alpha-\gamma/2}) = O(m_{\mathsf{SPRG}}^{2\alpha-\gamma})$. It follows by Lemma 4.5 that this can be done with a circuit of size $O(m_{\mathsf{SPRG}}^{2\alpha-\gamma/2})$.
- Calculating k from k' is a straightforward addition of integers that are at most L^2, so this is doable with a circuit of size $\mathsf{poly}(\log m_{\mathsf{SPRG}})$.
- The call to Lemma 4.7 takes $N = L^2 = O(m_{\mathsf{SPRG}}^{2\alpha})$, so it follows that this can be done with a circuit of size $O(L^{2(1-\gamma/(8\alpha))}) = O(m_{\mathsf{SPRG}}^{2\alpha-\gamma/4})$.
- Computing c and then $(x_i, y_i) = \psi(s_i)$ for all $i \in [c]$ is doable with a circuit of size $L' \cdot \mathsf{poly}(\log m_{\mathsf{SPRG}}) = O(m_{\mathsf{SPRG}}^{2\alpha-\gamma/2})$.
- Given an integer $j \in [L]$ as input, there exists a circuit of size $O(L \cdot \mathsf{poly}(\log L))$ to compute a vector in $\mathbb{Z}^L$ that is 1 in position j and 0 everywhere else. Indeed, this can be done on a two-tape Turing machine in $O(L \cdot \mathsf{poly}(\log L))$ time: one tape will store j and the second tape will store the output. The head on the second tape can pass over the output while decrementing the value on the first tape after each step, and write a 1 on the second tape if the first tape's value is exactly 0 and write a 0 otherwise. Our claim now follows from Lemma 4.4.

  It follows that assembling the matrices $\mathbf{U}_j$ and $\mathbf{V}_j$ is doable with a circuit of total size $O(L' \cdot L \cdot \mathsf{poly}(\log L)) = O(m_{\mathsf{SPRG}}^{3\alpha-\gamma} \cdot \mathsf{poly}(\log m_{\mathsf{SPRG}})) \leq O(m_{\mathsf{SPRG}}^{2\alpha-\gamma/5})$.

Hence the total circuit size for sampling $\mathbf{U}_j, \mathbf{V}_j$ for all $j \in [\ell]$ is $O(\ell \cdot m_{\mathsf{SPRG}}^{2\alpha-\gamma/5}) = O(m_{\mathsf{SPRG}}^{1-\gamma/5})$. Finally, sampling $\mathbf{r} \leftarrow U_n$ and adding that to the output is doable with a circuit of size $O(n) = O(m_{\mathsf{SPRG}}^{1/(1+\epsilon)}) = O(m_{\mathsf{SPRG}}^{1-\Omega(1)})$. Note importantly that the size of our circuit is determined by $\gamma = \delta/(1+\epsilon)$ and ϵ, which come directly from the parameters in the $\mathsf{SparseLPN}$ assumption and are hence independent of τ. This completes our proof that SdSamp is implementable with a sublinear-time circuit, and hence Theorem 4.1.

5 Combiner-Friendly Amortized Randomized Encodings

We use a very similar notion of an amortized randomized encoding (ARE) as in [JLS22], with the primary being how we instantiate the PRGs. The ARE construction of [JLS22] uses a PRG in NC^0 in two places:

1. A PRG in NC^0 with *polynomial* stretch as an initial step to generate, from a polynomially smaller seed, enough randomness for all of the wire labels and permutation bits in the garbling.

2. A PRG in NC^0 with *linear* stretch (specifically, length λ/B to $2\lambda+2$) to compute the garbled tables for each gate. We note that unlike the previous case, the evaluator needs to compute this PRG to decode.

To instantiate Item 1, we relax their construction by using our structured seed PRG instead of a PRG with polynomial stretch in NC^0. To instantiate Item 2, we use Corollary 3.5 to directly give a PRG (family) with linear stretch from the $\mathsf{SparseLPN}$ assumption.

A more minor difference from the construction by [JLS22] is that we require a notion of *combiner-friendliness* in our security definition. Specifically, we require the CFARE to work with B independently sampled crs strings, and retain security if at least one of the crs strings is secure. (Here, B is a universal constant; in fact, we will eventually take B to be 3.) Looking ahead, this will enable us to leverage the SKFE combiner by [JMS20] in Section 7.4 while preserving sublinear-time efficiency.

5.1 Definition

Since our structured seed PRG has an IdSamp phase, and since our linear stretch PRG is really a PRG family, we modify the definition of [JLS22] to allow a setup phase. Let $\mathcal{F}_{\mathsf{CFARE}} = \{\mathcal{F}_{\mathsf{CFARE},n_{\mathsf{CFARE}},m_{\mathsf{CFARE}},k_{\mathsf{CFARE}},\lambda}\}_{n_{\mathsf{CFARE}},m_{\mathsf{CFARE}}k_{\mathsf{CFARE}},\lambda\in\mathbb{N}}$ denote the set of all circuits $C : \{0,1\}^{n_{\mathsf{CFARE}}} \to \{0,1\}^{m_{\mathsf{CFARE}}\cdot k_{\mathsf{CFARE}}}$, where every output bit is computable by a circuit of size λ. Here, n_{CFARE}, m_{CFARE}, and k_{CFARE} are polynomials in λ, i.e., $\lambda^{\Theta(1)}$.

For simplicity, we abuse language and talk about the description length of a circuit and the size of a circuit interchangeably, both as λ in this case. These quantities are quasi-linearly related and will make no difference to any of our theorem statements.

Definition 13 (CFARE Syntax)**.** *A combiner-friendly* CFARE *scheme, parametrized by some* $B = O(1)$*, consists of the following p.p.t. algorithms:*

- $\mathsf{Setup}(1^{n_{\mathsf{CFARE}}}, 1^{m_{\mathsf{CFARE}}}, 1^{k_{\mathsf{CFARE}}}, 1^\lambda) \to \mathsf{crs}$. *The algorithm is randomized and outputs a public string* $\mathsf{crs} \in \{0,1\}^*$.
- $\mathsf{SdSamp}(1^{k_{\mathsf{CFARE}}}, 1^\lambda, n_{\mathsf{CFARE}}, m_{\mathsf{CFARE}}) \to \mathbf{r}$. *The algorithm is randomized and outputs a (private) binary string* $\mathbf{r} \in \{0,1\}^*$.
- $\mathsf{Encode}(C \in \mathcal{F}_{\mathsf{CFARE},n_{\mathsf{CFARE}},m_{\mathsf{CFARE}},k_{\mathsf{CFARE}},\lambda}, \mathbf{x} \in \{0,1\}^{n_{\mathsf{CFARE}}}, \{\mathsf{crs}_i\}_{i\in[B]}, \mathbf{r}) \to \mathbf{y}$. *The algorithm is* deterministic *and takes in a circuit* C *and input* $\mathbf{x}$ *and outputs some encoding* $\mathbf{y} \in \{0,1\}^*$.
- $\mathsf{Decode}(1^{n_{\mathsf{CFARE}}}, 1^{m_{\mathsf{CFARE}}}, 1^{k_{\mathsf{CFARE}}}, 1^\lambda, \mathbf{y}, \{\mathsf{crs}_i\}_{i\in[B]}) \to \mathbf{z}$. *The algorithm is deterministic and outputs some string* $\mathbf{z} \in \{\perp\} \cup \{0,1\}^{m_{\mathsf{CFARE}}\cdot k_{\mathsf{CFARE}}}$.

Definition 14 (CFARE Correctness)**.** *We say a* CFARE *scheme satisfies (perfect) correctness if for all circuits* $C \in \mathcal{F}_{\mathsf{CFARE},n_{\mathsf{CFARE}},m_{\mathsf{CFARE}},k_{\mathsf{CFARE}},\lambda}$ *and inputs* $\mathbf{x} \in \{0,1\}^{n_{\mathsf{CFARE}}}$,

$$\Pr\left[\mathbf{z} = C(\mathbf{x}) \;\middle|\; \begin{array}{c} \forall i \in [B] : \mathsf{crs}_i \leftarrow \mathsf{Setup}(1^{n_{\mathsf{CFARE}}}, 1^{m_{\mathsf{CFARE}}}, 1^{k_{\mathsf{CFARE}}}, 1^\lambda), \\ \mathbf{r} \leftarrow \mathsf{SdSamp}(1^{k_{\mathsf{CFARE}}}, 1^\lambda, n_{\mathsf{CFARE}}, m_{\mathsf{CFARE}}), \\ \mathbf{y} \leftarrow \mathsf{Encode}(C, \mathbf{x}, \{\mathsf{crs}_i\}_{i\in[B]}, \mathbf{r}), \\ \mathbf{z} \leftarrow \mathsf{Decode}(1^{n_{\mathsf{CFARE}}}, 1^{m_{\mathsf{CFARE}}}, 1^{k_{\mathsf{CFARE}}}, 1^\lambda, \mathbf{y}, \{\mathsf{crs}_i\}_{i\in[B]}) \end{array}\right] = 1.$$

Definition 15 (CFARE Indistinguishability Security)**.** *We say a* CFARE *scheme is* $\mu(\lambda)$*-secure if there exist distributions* $\mathsf{GoodSetup}(1^{n_{\mathsf{CFARE}}}, 1^{m_{\mathsf{CFARE}}}, 1^{k_{\mathsf{CFARE}}}, 1^\lambda)$ *and* $\mathsf{BadSetup}(1^{n_{\mathsf{CFARE}}}, 1^{m_{\mathsf{CFARE}}}, 1^{k_{\mathsf{CFARE}}}, 1^\lambda)$ *such that both of the following are true:*

1. *For all stateful p.p.t. adversaries $\mathcal{A}$ and strings* $\mathsf{good} \in \{0,1\}^B$ *which have* 1 *in at least one entry, there exists a negligible function* negl *such that $\mathcal{A}$ wins the following game with probability at most* $1/2 + \mathsf{negl}(\lambda)$*:*

    - *The adversary $\mathcal{A}$ takes as input 1^λ and chooses polynomial parameters $n_{\mathsf{CFARE}}, m_{\mathsf{CFARE}}, k_{\mathsf{CFARE}}$.*
    - *For each $j \in [B]$, the challenger samples* $\mathsf{crs}_j \leftarrow \mathsf{GoodSetup}(1^{n_{\mathsf{CFARE}}}, 1^{m_{\mathsf{CFARE}}}, 1^{k_{\mathsf{CFARE}}}, 1^\lambda)$ *if* $\mathsf{good}_j = 1$ *and* $\mathsf{crs}_j \leftarrow \mathsf{BadSetup}(1^{n_{\mathsf{CFARE}}}, 1^{m_{\mathsf{CFARE}}}, 1^{k_{\mathsf{CFARE}}}, 1^\lambda)$ *if* $\mathsf{good}_j = 0$*. The challenger then sends* $\{\mathsf{crs}_j\}_{j\in[B]}$ *to the adversary.*
    - *The adversary replies with a circuit* $C \in \mathcal{F}_{\mathsf{CFARE},n_{\mathsf{CFARE}},m_{\mathsf{CFARE}},k_{\mathsf{CFARE}},\lambda}$ *and inputs* $\mathbf{x}_0, \mathbf{x}_1 \in \{0,1\}^{n_{\mathsf{CFARE}}}$*. The challenger checks that* $C(\mathbf{x}_0) = C(\mathbf{x}_1)$*, and if not, rejects immediately.*
    - *The challenger samples* $b \leftarrow \{0,1\}$ *and replies with*

    $$\mathbf{y} \leftarrow \mathsf{Encode}\left(C, \mathbf{x}_b, \{\mathsf{crs}_j\}_{j\in[B]}, \mathbf{r} \leftarrow \mathsf{SdSamp}(1^{k_{\mathsf{CFARE}}}, 1^\lambda, n_{\mathsf{CFARE}}, m_{\mathsf{CFARE}})\right).$$

    - *The adversary outputs a guess* $b' \in \{0,1\}$ *and wins if and only if* $b = b'$*.*

2. *We have* $\mu = \Omega(1)$ *and the following two distributions are identical:*

    - *Sample* $\mathsf{crs} \leftarrow \mathsf{Setup}(1^{n_{\mathsf{CFARE}}}, 1^{m_{\mathsf{CFARE}}}, 1^{k_{\mathsf{CFARE}}}, 1^\lambda)$*.*
    - *With probability μ, sample* $\mathsf{crs} \leftarrow \mathsf{GoodSetup}(1^{n_{\mathsf{CFARE}}}, 1^{m_{\mathsf{CFARE}}}, 1^{k_{\mathsf{CFARE}}}, 1^\lambda)$*. With probability* $1-\mu$*, sample* $\mathsf{crs} \leftarrow \mathsf{BadSetup}(1^{n_{\mathsf{CFARE}}}, 1^{m_{\mathsf{CFARE}}}, 1^{k_{\mathsf{CFARE}}}, 1^\lambda)$*.*

If $\mathsf{negl}(\lambda) = \exp\left(-\lambda^{\Omega(1)}\right)$*, then we say* CFARE *is sub-exponentially μ-secure.*

Definition 16 (CFARE Sublinear Efficiency)**.** *For a* CFARE *scheme and* $c_3 > 0$*, we say that the scheme is c_3-sublinear if* $\mathsf{SdSamp}(1^{k_{\mathsf{CFARE}}}, 1^\lambda, n_{\mathsf{CFARE}}, m_{\mathsf{CFARE}})$ *is computable by a uniformly efficiently generatable randomized circuit of size at most* $O(k_{\mathsf{CFARE}} \cdot (n_{\mathsf{CFARE}} + m_{\mathsf{CFARE}}^{1-c_3}) \cdot \mathsf{poly}(\lambda))$*. In particular, the length of the output of* $\mathsf{SdSamp}(1^{k_{\mathsf{CFARE}}}, 1^\lambda, n_{\mathsf{CFARE}}, m_{\mathsf{CFARE}})$ *is at most* $O(k_{\mathsf{CFARE}} \cdot (n_{\mathsf{CFARE}} + m_{\mathsf{CFARE}}^{1-c_3}) \cdot \mathsf{poly}(\lambda))$*.*

Lastly, we define polynomial efficiency of a CFARE scheme.

Definition 17 (CFARE Polynomial Efficiency)**.** *For* $d \in \mathbb{N}$ *and* $c_1, c_2, \tau > 0$*, we say that a* CFARE *scheme is (d, c_1, c_2, τ)-efficient if the following holds. Given* $\mathsf{crs}_i \leftarrow \mathsf{Setup}(1^{n_{\mathsf{CFARE}}}, 1^{m_{\mathsf{CFARE}}}, 1^{k_{\mathsf{CFARE}}}, 1^\lambda)$ *for each $i \in [B]$, there exists an efficiently sampleable degree-d monomial pattern* $\mathcal{Q}_{\{\mathsf{crs}_i\}_{i\in[B]}}$ *of size* $m'_{\mathsf{CFARE}} = O((n_{\mathsf{CFARE}} + m_{\mathsf{CFARE}})^{1+\tau}\lambda^{c_1})$ *such that for any circuit* $C \in \mathcal{F}_{\mathsf{CFARE},n_{\mathsf{CFARE}},m_{\mathsf{CFARE}},k_{\mathsf{CFARE}},\lambda}$ *and input* $\mathbf{x} \in \{0,1\}^{n_{\mathsf{CFARE}}}$*, the mapping* $\mathsf{Encode}(C, \mathbf{x}, \{\mathsf{crs}_i\}_{i\in[B]}, \mathbf{r}) \rightarrow \mathbf{y}$ *satisfies the following requirements:*

- *For* $\mathbf{r} = (\mathbf{r}_1, \cdots, \mathbf{r}_{k_{\mathsf{CFARE}}})$ *where each $\mathbf{r}_i$ has equal size, let* $\mathbf{a}_i = (\mathbf{x}, \mathbf{r}_i) \in \{0,1\}^*$*. The length of each $\mathbf{a}_i$ is* $n'_{\mathsf{CFARE}} = O((n_{\mathsf{CFARE}} + m_{\mathsf{CFARE}}^{1-c_2})\lambda^{c_1})$*.*
- *For all $i \in |\mathbf{y}|$, there exists efficiently sampleable* $\mu_{i,Q,j} \in \mathbb{Z}$ *given* crs *such that for all* $\mathbf{x} \in \{0,1\}^{n_{\mathsf{CFARE}}}$ *and* $\mathbf{r} \in \{0,1\}^{k_{\mathsf{CFARE}} \cdot n'_{\mathsf{CFARE}}}$*,*

$$y_i = \sum_{Q \in \mathcal{Q}_{\{\mathsf{crs}_i\}_{i\in[B]}}, j \in [k_{\mathsf{CFARE}}]} \mu_{i,Q,j} \cdot \mathsf{Mon}_Q(\mathbf{a}_j).$$

Note that restricting these polynomial representations of y_i to be multilinear in $\mathbf{a}_j$ is without loss of generality, since $\mathbf{a}_j \in \{0,1\}^*$.

We now state our main theorem of this section.

Theorem 5.1. *Suppose there exist constants $d \in \mathbb{N}$ and $\nu > 0$ such that for all $\tau > 0$, there exists a perfectly correct and (sub-exponentially) μ-secure* SPRG *with polynomial stretch satisfying ν-sublinear efficiency with degree d and τ-local decompression. Suppose also that there exists a (sub-exponentially) secure linear-stretch* PRG *family in* NC^0 *(per Definition 5).*

Then for any $B = O(1)$, there exist constants $d' \in \mathbb{N}, c_1, c_2, c_3 > 0$ such that for all $\tau > 0$, there exists a perfectly correct, (sub-exponentially) μ^2-secure, c_3-sublinear, and (d', c_1, c_2, τ)-efficient CFARE.

Since we have shown that the $\mathsf{SparseLPN}$ assumption satisfies both pre-conditions, we have the following corollary:

Corollary 5.2. *Suppose that the (sub-exponential)* $\mathsf{SparseLPN}$ *assumption holds. Then for any $B = O(1)$, there exist constants $d \in \mathbb{N}$ and $c_1, c_2, c_3 > 0$ such that for all $\tau > 0$, there exists a perfectly correct, (sub-exponentially) $\Omega(1)$-secure, c_3-sublinear, and (d, c_1, c_2, τ)-efficient* CFARE.

Proof. This follows immediately from combining Theorem 5.1, Theorem 4.1, and Corollary 3.5. □

5.2 Construction

We define some useful notation to be used throughout the construction. Recall that n'_{CFARE} is the length of each $\mathbf{a}_i = (\mathbf{x}, \mathbf{r}_i)$. Let $U = U_{m_{\mathsf{CFARE}} \cdot \lambda, n_{\mathsf{CFARE}}, m_{\mathsf{CFARE}}} : \{0,1\}^{m_{\mathsf{CFARE}} \cdot \lambda} \times \{0,1\}^{n_{\mathsf{CFARE}}} \to \{0,1\}^{m_{\mathsf{CFARE}}}$ denote the universal circuit corresponding to evaluating a circuit C with n_{CFARE}-length inputs, m_{CFARE}-length outputs, and total description length $m_{\mathsf{CFARE}} \cdot \lambda$. That is, for all such C and inputs $\mathbf{x} \in \{0,1\}^n$, we have $U(C, \mathbf{x}) = C(\mathbf{x})$.

We will use a linear stretch PRG family and a SPRG with polynomial stretch with the following parameters. For PRG, we will set $n_{\mathsf{PRG}} := \lambda/B$ with output length $2\lambda + 2 = n_{\mathsf{PRG}} \cdot 2B + 2 = O(n_{\mathsf{PRG}})$. For SPRG, we set $m_{\mathsf{SPRG}} := (n_{\mathsf{CFARE}} + m_{\mathsf{CFARE}}) \cdot \mathsf{poly}(\lambda)$.

CFARE Construction (Part 1)

$\mathsf{CFARE.Setup}(1^{n_{\mathsf{CFARE}}}, 1^{m_{\mathsf{CFARE}}}, 1^{k_{\mathsf{CFARE}}}, 1^{\lambda})$:

1. Output $\mathsf{crs} = (I_{\mathsf{PRG}} \leftarrow \mathsf{PRG.IdSamp}(1^{n_{\mathsf{PRG}}}), I_{\mathsf{SPRG}} \leftarrow \mathsf{SPRG.IdSamp}(1^{m_{\mathsf{SPRG}}}))$.

$\mathsf{CFARE.SdSamp}(1^{k_{\mathsf{CFARE}}}, 1^{\lambda}, n_{\mathsf{CFARE}}, m_{\mathsf{CFARE}})$:

1. Run $\mathsf{SPRG.SdSamp}(m_{\mathsf{SPRG}})$ independently $B \cdot k_{\mathsf{CFARE}}$ times to get $\mathbf{r}_{i,j}$ for each $i \in [k_{\mathsf{CFARE}}]$ and $j \in [B]$, and output $\mathbf{r} = \{\mathbf{r}_{i,j}\}_{i \in [k_{\mathsf{CFARE}}], j \in [B]}$.

$\mathsf{CFARE.Decode}(1^{n_{\mathsf{CFARE}}}, 1^{m_{\mathsf{CFARE}}}, 1^{k_{\mathsf{CFARE}}}, 1^{\lambda}, \mathbf{y}, \{\mathsf{crs}_j\}_{j \in [B]})$:

1. Parse $\mathbf{y} = (\mathbf{y}_1, \cdots, \mathbf{y}_{k_{\mathsf{CFARE}}})$.
2. For each $\kappa \in [k_{\mathsf{CFARE}}]$, run Yao's garbled circuit evaluation on $\mathbf{y}_\kappa$ using $\{I_{\mathsf{PRG},j}\}_{j \in [B]}$ to generate output $\mathbf{z}_\kappa \in \{0,1\}^{m_{\mathsf{CFARE}}}$, and finally output $\mathbf{z} = (\mathbf{z}_1, \cdots, \mathbf{z}_{k_{\mathsf{CFARE}}})$.

Figure 4: Our CFARE Construction (except $\mathsf{CFARE.Encode}$). See Figure 5 for the rest of the construction.

CFARE Construction (Part 2)

$\mathsf{CFARE.Encode}(C, \mathbf{x} \in \{0,1\}^{n_{\mathsf{CFARE}}}, \{\mathsf{crs}_j\}_{j\in[B]}, \mathbf{r})$:

1. Write $C = (C_1, \cdots, C_{k_{\mathsf{CFARE}}})$, where each $C_i : \{0,1\}^{n_{\mathsf{CFARE}}} \to \{0,1\}^{m_{\mathsf{CFARE}}}$ outputs the ith output chunk of C. The description length of each C_i is $m_{\mathsf{CFARE}} \cdot \lambda$. Parse $\mathbf{r} = \{\mathbf{r}_{i,j}\}_{i\in[k_{\mathsf{CFARE}}], j\in[B]}$, and set $\mathbf{a}_i = (\mathbf{x}, \mathbf{r}_i := \{\mathbf{r}_{i,j}\}_{j\in[B]})$ for all $i \in [k_{\mathsf{CFARE}}]$. Also, parse $\mathsf{crs}_j = (I_{\mathsf{PRG},j}, I_{\mathsf{SPRG},j})$ for each $j \in [B]$.

2. For all $\kappa \in [k_{\mathsf{CFARE}}]$:

   - Evaluate $\bigoplus_{j\in[B]} \mathsf{SPRG.Eval}(I_{\mathsf{SPRG},j}, \mathbf{r}_{\kappa,j})$ to generate $\boldsymbol{\sigma}||\mathbf{b}$ of length $m_{\mathsf{SPRG}} = (n_{\mathsf{CFARE}} + m_{\mathsf{CFARE}}) \cdot \mathsf{poly}(\lambda)$. We will use $\boldsymbol{\sigma}$ for the garbling and $\mathbf{b}$ for the permutation bits. For each wire w in U, we let $\boldsymbol{\sigma}_{w,0}, \boldsymbol{\sigma}_{w,1} \in \{0,1\}^\lambda$ be the two labels for the wire, and let $b_w \in \{0,1\}$ be the permutation bit for the wire.

   - Generate input labels of $(C_\kappa, \mathbf{x})$ as follows. For every input wire $w_{\mathsf{ckt},i}$ for $i \in [m_{\mathsf{CFARE}} \cdot \lambda]$ and $w_{\mathsf{inp},j}$ for $j \in [n_{\mathsf{CFARE}}]$, set

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

     where $C_{\kappa,i}$ is the ith bit of the description of C_κ.

   - Now, for any string $\mathbf{s} = (\mathbf{s}_1, \ldots, \mathbf{s}_B) \in \{0,1\}^\lambda$, we can write $\bigoplus_{j\in[B]} \mathsf{PRG.Eval}(I_{\mathsf{PRG},j}, \mathbf{s}_j) = \mathsf{G}_0(\mathbf{s})||\mathsf{G}_1(\mathbf{s})$, where $\mathsf{G}_0, \mathsf{G}_1 : \{0,1\}^\lambda \to \{0,1\}^{\lambda+1}$. For every gate gate in U computing a function $g : \{0,1\}^2 \to \{0,1\}$ with input wires w_1, w_2 and output wire w_3, set

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

   - Let $w_{\mathsf{out},\gamma}$ for $\gamma \in [m_{\mathsf{CFARE}}]$ denote output wire γ. For $\gamma \in [m_{\mathsf{CFARE}}]$, set $\mathsf{OutTab}_\gamma = ((0, \boldsymbol{\sigma}_{w_{\mathsf{out},\gamma},0}), (1, \boldsymbol{\sigma}_{w_{\mathsf{out},\gamma},1}))$.

   - Set

$$\mathbf{y}_\kappa = \left(\{\mathsf{Lab}_{C_\kappa,i}\}_{i\in[m_{\mathsf{CFARE}}\cdot\lambda]}, \{\mathsf{Lab}_j\}_{j\in[n_{\mathsf{CFARE}}]}, \{T_{\mathsf{gate}}\}_{\mathsf{gate}\in\mathsf{gate}(U)}, \{\mathsf{OutTab}_\gamma\}_{\gamma\in[m_{\mathsf{CFARE}}]}\right).$$

3. Output $\mathbf{y} = \{\mathbf{y}_\kappa\}_{\kappa\in[k_{\mathsf{CFARE}}]}$.

Figure 5: Our CFARE Construction (only CFARE.Encode). See Figure 4 for the rest of the construction.

We give the construction in Figures 4 and 5.

We now discuss why this construction proves Theorem 5.1.

Correctness: Correctness immediately holds by construction of Yao's garbling scheme [Yao86, BMR90].

Indistinguishability Security: Security holds readily assuming the pseudorandomness of PRG and SPRG due to the security of Yao's garbling scheme [Yao86, BMR90, LP04]. Let $\mu = \Omega(1)$ be such that $\mathsf{PRG}, \mathsf{SPRG}$ are both μ-secure. We will take $\mathsf{GoodSetup}$ to sample $I_{\mathsf{PRG}} \leftarrow \mathsf{PRG.GoodIdSamp}(1^{n_{\mathsf{PRG}}})$ and $I_{\mathsf{SPRG}} \leftarrow \mathsf{SPRG.GoodIdSamp}(1^{m_{\mathsf{SPRG}}})$. We will take $\mathsf{BadSetup}$ to sample from the following mixture:

- With probability proportional $\mu(1-\mu)$, sample $I_{\mathsf{PRG}} \leftarrow \mathsf{PRG.BadIdSamp}(1^{n_{\mathsf{PRG}}})$ and $I_{\mathsf{SPRG}} \leftarrow \mathsf{SPRG.GoodIdSamp}(1^{m_{\mathsf{SPRG}}})$.
- With probability proportional to $\mu(1-\mu)$, sample $I_{\mathsf{PRG}} \leftarrow \mathsf{PRG.GoodIdSamp}(1^{n_{\mathsf{PRG}}})$ and $I_{\mathsf{SPRG}} \leftarrow \mathsf{SPRG.BadIdSamp}(1^{m_{\mathsf{SPRG}}})$.
- With probability proportional to $(1-\mu)^2$, sample $I_{\mathsf{PRG}} \leftarrow \mathsf{PRG.BadIdSamp}(1^{n_{\mathsf{PRG}}})$ and $I_{\mathsf{SPRG}} \leftarrow \mathsf{SPRG.BadIdSamp}(1^{m_{\mathsf{SPRG}}})$.

Then it is clear that sampling from Setup is equivalent to sampling from $\mathsf{GoodSetup}$ with probability $\mu^2 = \Omega(1)$ and from $\mathsf{BadSetup}$ with probability $1-\mu^2$. It remains to check indistinguishability security for this characterization of $\mathsf{GoodSetup}$ and $\mathsf{BadSetup}$.

Let $\{I_{\mathsf{SPRG},j}\}_{j\in[B]}$ be the j SPRG function indices that we sample for the CFARE. There exists some $j^* \in [B]$ such that $\mathsf{good}_{j^*} = 1$, so that I_{SPRG,j^*} will be a sample from $\mathsf{SPRG.GoodIdSamp}(1^{m_{\mathsf{SPRG}}})$. For each $\kappa \in [k_{\mathsf{CFARE}}]$, it hence follows by the security of the SPRG that $(I_{\mathsf{SPRG},j^*}, \mathsf{SPRG.Eval}(I_{\mathsf{SPRG},j^*}, \mathbf{r}_{\kappa,j^*}))$ is computationally indistinguishable from $(I_{\mathsf{SPRG},j^*}, U_{m_{\mathsf{SPRG}}})$, with security loss $\mathsf{negl}(m_{\mathsf{SPRG}}) = \mathsf{negl}((n_{\mathsf{CFARE}} + m_{\mathsf{CFARE}}) \cdot \mathsf{poly}(\lambda)) \leq \mathsf{negl}(\lambda)$. It then follows that

$$\left(\{I_{\mathsf{SPRG},j}\}_{j\in[B]}, \bigoplus_{j\in[B]} \mathsf{SPRG.Eval}(I_{\mathsf{SPRG},j}, \mathbf{r}_{\kappa,j}) \right) \approx_c \left(\{I_{\mathsf{SPRG},j}\}_{j\in[B]}, U_{m_{\mathsf{SPRG}}} \right).$$

This is because the SPRG seeds are sampled independently.

A similar argument also applies for the function mapping $\mathbf{s} \mapsto \bigoplus_{j\in[B]} \mathsf{PRG.Eval}(I_{\mathsf{PRG},j}, \mathbf{s}_j)$; this will matter in the next step where we use the security of Yao's garbled circuits. Note also that the number of invocations of $\mathsf{SPRG.Eval}$ and $\mathsf{PRG.Eval}$ is bounded above by $\mathsf{poly}(\lambda)$, so the total security loss will also be $\mathsf{negl}(\lambda)$ (with sub-exponential indistinguishability if the PRG and SPRG indistinguishabilities are also sub-exponential). We remark that this argument holds even though the adversary can select $C, \mathbf{x}_0, \mathbf{x}_1$ adaptively based on $\{\mathsf{crs}_j\}_{j\in[B]}$, as the PRG and SPRG security properties are completely non-interactive.

It hence follows that producing an encoding for $\mathbf{x}_b$ using our SPRG outputs is computationally indistinguishable from producing an encoding for $\mathbf{x}_b$ using honest random bits. Security of Yao's garbled circuits [Yao86, BMR90, LP04] (together with our above observation about the calls to $\mathsf{PRG.Eval}$) now implies that provided $C(\mathbf{x}_0) = C(\mathbf{x}_1)$, an encoding for $\mathbf{x}_0$ will be computationally indistinguishable from an encoding of $\mathbf{x}_1$.

Remark. We remark that in the above analysis, $\{I_{\mathsf{SPRG},j}\}_{j\neq j^*}$ may not be efficiently sampleable (since they come from $\mathsf{SPRG.GoodIdSamp}$ or $\mathsf{SPRG.BadIdSamp}$); this is not a problem for us since we can have a non-uniform adversary for the SPRG that takes $\{I_{\mathsf{SPRG},j}\}_{j\neq j^*}$ as advice. Throughout this paper, we will repeatedly use this type of argument to handle the (possibly) inefficient distributions we sample from.

For the purposes of analyzing efficiency, recall the formulation of Theorem 4.1 that tells us there exist $d \in \mathbb{N}$ and $\nu > 0$ such that for any $\tau > 0$, there exists a SPRG with ν-sublinearity and degree d and τ-local decompression. We consider our CFARE construction instantiated with such a SPRG in the below analysis.

Sublinear Efficiency: We observe that the circuit size of $\mathsf{CFARE.SdSamp}$ can directly be upper bounded by $O(B \cdot k_{\mathsf{CFARE}} \cdot m_{\mathsf{SPRG}}^{1-\nu})$, since sublinear efficiency of SPRG guarantees that $\mathsf{SPRG.SdSamp}$ has circuit size $O(m_{\mathsf{SPRG}}^{1-\nu})$. Expanding this out and noting that $B = O(1)$, we have an upper bound of

$$\begin{aligned} k_{\mathsf{CFARE}} \cdot m_{\mathsf{SPRG}}^{1-\nu} &= k_{\mathsf{CFARE}} \cdot ((n_{\mathsf{CFARE}} + m_{\mathsf{CFARE}}) \cdot \mathsf{poly}(\lambda))^{1-\nu} \\ &\leq k_{\mathsf{CFARE}} \cdot \left(n_{\mathsf{CFARE}} + m_{\mathsf{CFARE}}^{1-\nu}\right) \cdot \mathsf{poly}(\lambda), \end{aligned}$$

as desired.

Polynomial Efficiency: We first analyze n'_{CFARE}, the size of each $\mathbf{a}_i = (\mathbf{x}, \mathbf{r}_i = \{\mathbf{r}_{i,j}\}_{j\in[B]})$ (for $i \in [k_{\mathsf{CFARE}}]$). Clearly $|\mathbf{x}| = n_{\mathsf{CFARE}}$ by definition, and $|\mathbf{r}_{i,j}|$ is upper-bounded by the circuit size of $\mathsf{SPRG.SdSamp}(m_{\mathsf{SPRG}})$. Recall that by sublinear efficiency of SPRG, $\mathsf{SPRG.SdSamp}$ has circuit size $O(m_{\mathsf{SPRG}}^{1-\nu})$, and since $m_{\mathsf{SPRG}} = (n_{\mathsf{CFARE}} + m_{\mathsf{CFARE}}) \cdot \mathsf{poly}(\lambda)$, we have

$$n'_{\mathsf{CFARE}} = |\mathbf{a}_i| \leq n_{\mathsf{CFARE}} + ((n_{\mathsf{CFARE}} + m_{\mathsf{CFARE}}) \cdot \mathsf{poly}(\lambda))^{1-\nu} \leq (n_{\mathsf{CFARE}} + m_{\mathsf{CFARE}}^{1-\nu}) \cdot \mathsf{poly}(\lambda),$$

as desired.

We next analyze the polynomial representation of the mapping that takes an arbitrary $\mathbf{a}_\kappa = (\mathbf{x}, \mathbf{r}_\kappa = \{\mathbf{r}_{\kappa,j}\}_{j\in[B]})$ to any individual output bit of the garbling $i \in |\mathbf{y}|$, where $\mathbf{y}$ is the output of $\mathsf{CFARE.Encode}$. Recall that $\mathbf{y} = \{\mathbf{y}_\kappa\}_{\kappa\in[k_{\mathsf{CFARE}}]}$, so depending on which of the k_{CFARE} chunks i is in, there exists a unique index $\kappa \in [k_{\mathsf{CFARE}}]$ for which y_i depends on $\mathbf{a}_\kappa$. We can write this mapping as a composition of two functions

$$\mathbf{a}_\kappa = (\mathbf{x}, \{\mathbf{r}_{\kappa,j}\}_{j\in[B]}) \overset{g_1}{\longmapsto} (\mathbf{x}, \boldsymbol{\sigma}||\mathbf{b}) \overset{g_2}{\longmapsto} \mathbf{y}_\kappa.$$

Here, g_1 maps $(\mathbf{x}, \{\mathbf{r}_{\kappa,j}\}_{j\in[B]}) \mapsto (\mathbf{x}, \bigoplus_{j\in[B]} \mathsf{SPRG.Eval}(I_{\mathsf{SPRG},j}, \mathbf{r}_{\kappa,j}))$. We emphasize that the descriptions of the functions g_1 and g_2 do *not* depend on $\kappa \in [k_{\mathsf{CFARE}}]$ (in particular, because the $I_{\mathsf{SPRG},j}$'s and $I_{\mathsf{PRG},j}$'s are shared across all k_{CFARE} chunks), except for the dependence on the circuit chunk C_κ, which only affects the coefficients of the input label monomials corresponding to the circuit chunk.

We can further decompose g_1 into $g_{1,1} \circ g_{1,0}$, where $g_{1,0}(\mathbf{x}, \{\mathbf{r}_{\kappa,j}\}_{j\in[B]}) = (\mathbf{x}, \{\mathsf{SPRG.Eval}(I_{\mathsf{SPRG},j}, \mathbf{r}_{\kappa,j})\}_{j\in[B]})$ and $g_{1,1}(\mathbf{x}, \{\mathbf{v}_j\}_{j\in[B]}) = (\mathbf{x}, \bigoplus_{j\in[B]} \mathbf{v}_j)$. By the efficiency properties of the SPRG, we have $\deg(g_{1,0}) \leq d$ and $\mathsf{loc}(g_{1,0}) = O(m_{\mathsf{SPRG}}^{\tau})$. Moreover, $g_{1,1}$ has locality B so by considering the multilinear representation of Boolean functions in NC^0, we have $\deg(g_{1,1}) \leq B$ and $\mathsf{loc}(g_{1,1}) \leq B$. Moreover, we have $\mathsf{loc}(g_2) = O(1) \Rightarrow \deg(g_2) = O(1)$ independent of τ, since $\mathsf{PRG.Eval}(I_{\mathsf{PRG},j}, \cdot)$ is local for all $j \in [B]$ and all operations are local (including the bitwise XOR when computing $\bigoplus_{j\in[B]} \mathsf{PRG.Eval}(I_{\mathsf{PRG},j}, \mathbf{s}_j)$). Let $f = g_2 \circ g_{1,1} \circ g_{1,0}$ be the composition of g_2 and g_1, denoting the mapping from $\mathbf{a}_\kappa$ to $\mathbf{y}_\kappa$. By Lemmas 3.1 and 3.2, it follows that $\mathsf{loc}(f) = O(m_{\mathsf{SPRG}}^{\tau})$ and $\deg(f) = O(1)$ independent of τ. By Lemma 3.3, it follows that f can represented as a linear combination of

$$1 + \mathsf{loc}(f)^{\deg(f)} \leq m_{\mathsf{SPRG}}^{O(\tau)} = ((n_{\mathsf{CFARE}} + m_{\mathsf{CFARE}}) \cdot \mathsf{poly}(\lambda))^{O(\tau)} \leq (n_{\mathsf{CFARE}} + m_{\mathsf{CFARE}})^{O(\tau)} \cdot \mathsf{poly}(\lambda)$$

multilinear monomials. Now, since these monomials are independent of κ, by union bounding over just the $(n_{\mathsf{CFARE}} + m_{\mathsf{CFARE}}) \cdot \mathsf{poly}(\lambda)$ output bits $\mathbf{y}_\kappa$, we know there is a fixed monomial degree-$O(1)$ monomial pattern $\mathcal{Q}_{\{\mathsf{crs}_j\}_{j\in[B]}}$ of size

$$m'_{\mathsf{CFARE}} = (n_{\mathsf{CFARE}} + m_{\mathsf{CFARE}}) \cdot \mathsf{poly}(\lambda) \cdot (n_{\mathsf{CFARE}} + m_{\mathsf{CFARE}})^{O(\tau)} \cdot \mathsf{poly}(\lambda) = (n_{\mathsf{CFARE}} + m_{\mathsf{CFARE}})^{1+O(\tau)} \cdot \mathsf{poly}(\lambda),$$

as desired. Moreover, all steps in generating this multilinear polynomial representation are efficiently computable given $\{\mathsf{crs}_j\}_{j\in[B]}$.

We also point out that all parts of the output are independent of the input circuit C, except for the input label. However, as [JLS22] points out, these dependencies on the input circuit can be subsumed into the coefficients of the polynomial, making the monomial pattern independent of the circuit.

In summary, we have constructed an CFARE that that is ν-sublinear and $(d', c_1, \nu, O(\tau))$-efficient, where $d' \in \mathbb{N}$ and $c_1, \nu > 0$ can all be specified independently of τ. This establishes Theorem 5.1.

6 Combiner-Friendly Preprocessed Randomized Encodings

We will use the notion of a preprocessed randomized encoding (PRE) scheme, as defined by Jain, Lin, and Sahai [JLS22], but minor modifications to make it combiner-friendly. The purpose of the PRE is that it is made to convert partially hiding functional encryption (as in [Wee20]) into general purpose functional encryption, which can then be bootstrapped into indistinguishability obfuscation.

6.1 Ingredient: Preprocessed Polynomial Encodings

Here, we define the notion of a Preprocessed Polynomial Encoding (PPE) scheme, as defined in [JLS22]. This will be an ingredient in our CFPRE construction.

Definition 18 (Definition 4.2 of [JLS22])**.** *For a constant $d \in \mathbb{N}$, the family of classes of polynomials $\mathcal{F}_{\mathsf{PPE},d} = \{\mathcal{F}_{\mathsf{PPE},d,n_{\mathsf{PPE}},\mathcal{Q},k_{\mathsf{PPE}}}\}_{n\geq d_{\mathsf{PPE}},\mathcal{Q}\in\Gamma_{d,n_{\mathsf{PPE}}},k_{\mathsf{PPE}}\in\mathbb{N}}$ 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 $\left(\zeta_i^{(j)}\right)_{j\in[k_{\mathsf{PPE}}],i\in[m_{\mathsf{PPE}}]}$. For an input $\mathbf{x}$ consisting of k_{PPE} blocks $\mathbf{x} = (\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(k_{\mathsf{PPE}})})$, each of length n, f has the form*

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

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

Roughly speaking, $\mathcal{F}_{\mathsf{PPE},d}$ consists of polynomials that take as input k_{PPE} blocks of size n_{PPE}, and computes some linear combination of a fixed set of m_{PPE} degree-d monomials on inputs governed by a set $\mathcal{Q}$. Later, we will require that the size of the circuit computing $(\mathsf{PI}, \mathsf{SI})$ is sublinear in $|\mathcal{Q}| \cdot k_{\mathsf{PPE}}$.

Definition 19 (Syntax of PPE, Definition 4.3 of [JLS22])**.** *For any constant $d \in \mathbb{N}$, a PPE scheme for $\mathcal{F}_{\mathsf{PPE},d}$ 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 algorithm is randomized, and takes in the block length n_{PPE}, number of blocks k_{PPE}, a prime number p, a d-monomial pattern over n variables $\mathcal{Q}$ of size m_{PPE}, and an input $\mathbf{x} \in \mathbb{Z}_p^{n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}}$. It outputs a public string PI and a private string SI, both vectors 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 algorithm is deterministic. It takes as input a description of $f \in \mathcal{F}_{\mathsf{PPE},d,n_{\mathsf{PPE}},\mathcal{Q},k_{\mathsf{PPE}}}$ and a pre-processed input $(\mathsf{PI}, \mathsf{SI})$, and it outputs $y \in \mathbb{Z}_p$.*

We now define the correctness, security, and efficiency constraints on PPE, just as in [JLS22].

Definition 20 (PPE Correctness, Definition 4.4 of [JLS22])**.** *For a constant $d \in \mathbb{N}$, a PPE scheme is* correct *for the function class $\mathcal{F}_{d,\mathsf{PPE}}$ if the following holds. For all $k_{\mathsf{PPE}} \in \mathbb{N}, n_{\mathsf{PPE}} = k^{\Theta(1)}$, and $\mathcal{Q} \in \Gamma_{d,n_{\mathsf{PPE}}}$ with $m_{\mathsf{PPE}} \geq 1$, and for all $f \in \mathcal{F}_{\mathsf{PPE},d,n_{\mathsf{PPE}},\mathcal{Q},k_{\mathsf{PPE}}}$, primes p, and inputs $\mathbf{x} \in \mathbb{Z}_p^{n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}}$,*

$$\Pr\Big[\mathsf{Eval}(f, (\mathsf{PI}, \mathsf{SI})) = f(\mathbf{x}) \mod p \mid (\mathsf{PI}, \mathsf{SI}) \leftarrow \mathsf{PreProc}(1^{n_{\mathsf{PPE}}}, 1^{k_{\mathsf{PPE}}}, p, \mathcal{Q}, \mathbf{x})\Big] = 1.$$

Remark. Definition 4.4 in [JLS22] gives an *imperfect, statistical* correctness definition, where the probability of being correct is $1 - O(\exp(-k_{\mathsf{PPE}}^{\Omega(1)}))$ instead of 1. One can easily modify their construction to achieve *perfect* correctness, as in our definition, by simply outputting the value of $\mathbf{x}$ in the flag cases in the clear in PI, at the cost of an additive $\exp(-k_{\mathsf{PPE}}^{\Omega(1)})$ loss in security, which is tolerable.

Definition 21 (PPE Security, Definition 4.5 of [JLS22])**.** *Let $d \in \mathbb{N}$ be a constant. A PPE scheme is (sub-exponentially) 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 polynomially bounded function of 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 $\mathbf{x}_{k_{\mathsf{PPE}}} \in \mathbb{Z}_p^{n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}}$, and let $\{\mathcal{Q}_{k_{\mathsf{PPE}}}\}_{k_{\mathsf{PPE}} \in \mathbb{N}}$ be an ensemble of monomial patterns $\mathcal{Q}_{k_{\mathsf{PPE}}} \in \Gamma_{d,n_{\mathsf{PPE}}}$ with size $m_{\mathsf{PPE}} \geq 1$. Then, the following distributions are (sub-exponentially) indistinguishable:*

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

Definition 22 (Sublinear Efficiency of PPE, Definition 4.6 of [JLS22])**.** *Let $d \in \mathbb{N}$ be constant. We say that a PPE scheme for $\mathcal{F}_{\mathsf{PPE},d}$ satisfies sublinear efficiency if there exist constants c_1, c_2, c_3 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 prime p, the size of the (randomized) circuit computing $\mathsf{PreProc}(1^{n_{\mathsf{PPE}}}, 1^{k_{\mathsf{PPE}}}, p, \mathcal{Q}, \cdot)$ is $O((n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}^{c_1} + m_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}^{1-c_2}) \cdot (\log_2 p)^{c_3})$.*

Moreover, this circuit should be uniformly efficiently generatable.

Remark. Definition 4.6 in [JLS22] also includes a $\mathsf{poly}(k_{\mathsf{PPE}}) \cdot \mathsf{poly}(\log p)$ term in the circuit size of $\mathsf{PreProc}$, but this is dominated by the $n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}^{c_1} \cdot (\log_2 p)^{c_3}$ term.

Definition 23 (Complexity of PPE Evaluation, Definition 4.7 of [JLS22])**.** *Let $d \in \mathbb{N}$ be a constant. We require that a* PPE *scheme for $\mathcal{F}_{\mathsf{PPE},d}$ satisfy the following. For every $k_{\mathsf{PPE}} \in \mathbb{N}$, $n_{\mathsf{PPE}} = k_{\mathsf{PPE}}^{\Theta(1)}$, any $\mathcal{Q} \in \Gamma_{d,n_{\mathsf{PPE}}}$ of size $m_{\mathsf{PPE}} \geq 1$, any prime p, and any $f \in \mathcal{F}_{\mathsf{PPE},d,n_{\mathsf{PPE}},\mathcal{Q},k_{\mathsf{PPE}}}$, there exists an efficiently computable multivariate polynomial $g_{f,\mathcal{Q}}(\cdot,\cdot)$ (given the description of $f, \mathcal{Q}$) over $\mathbb{Z}_p$ of total degree $O(d)$ in* PI *and total degree 2 in* SI *such that for all inputs $\mathbf{x} \in \mathbb{Z}_p^{n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}}$ and all* (PI, SI) *in the support of $\mathsf{PreProc}(1^{n_{\mathsf{PPE}}}, 1^{k_{\mathsf{PPE}}}, p, \mathcal{Q}, \mathbf{x})$,*

$$f(\mathbf{x}) = \mathsf{Eval}(f, (\mathsf{PI}, \mathsf{SI})) = g_{f,\mathcal{Q}}(\mathsf{PI}, \mathsf{SI}) \pmod p.$$

Finally, we state a theorem we use directly from [JLS22], which states that PPE exists if LPN holds over large fields.

Theorem 6.1 (Theorem 4.2 of [JLS22])**.** *Assuming* LPN *over large fields (as in Definition 6), for all constants $d \in \mathbb{N}$, there exists a* PPE *scheme for $\mathcal{F}_{\mathsf{PPE},d}$ satisfying all requirements. Moreover, assuming sub-exponential* LPN *over large fields, for all constants $d \in \mathbb{N}$, there is a sub-exponentially secure* PPE *scheme for $\mathcal{F}_{\mathsf{PPE},d}$.*

6.2 Definition of CFPRE

As in Section 5.1, we modify the definition of [JLS22] to accommodate a setup phase and be combiner-friendly. Let $\mathcal{F}_{\mathsf{CFPRE}} = \{\mathcal{F}_{\mathsf{CFPRE},n_{\mathsf{CFPRE}},m_{\mathsf{CFPRE}},k_{\mathsf{CFPRE}},\lambda}\}_{n_{\mathsf{CFPRE}},m_{\mathsf{CFPRE}},k_{\mathsf{CFPRE}},\lambda\in\mathbb{N}}$ denote the set of all circuits $C : \{0,1\}^{n_{\mathsf{CFPRE}}} \to \{0,1\}^{m_{\mathsf{CFPRE}} \cdot k_{\mathsf{CFPRE}}}$ where every output bit is computable by a circuit of size λ. Here, $m_{\mathsf{CFPRE}}, n_{\mathsf{CFPRE}}, k_{\mathsf{CFPRE}}$ are polynomials in λ (i.e., $\lambda^{\Theta(1)}$), where we need the additional constraint that k_{CFPRE} is lower-bounded by a fixed polynomial in λ, and n_{CFPRE} and m_{CFPRE} are upper-bounded by a fixed polynomial in k_{CFPRE}.

Remark. The reason we need k_{CFPRE} to be lower-bounded by a fixed polynomial in λ is to get fixed sub-exponential indistinguishability when using PPE, whose security is parameterized by $k_{\mathsf{PPE}} = k_{\mathsf{CFPRE}}$. Similarly, we need n_{CFPRE} and m_{CFPRE} to be upper-bounded by a fixed polynomial in k_{CFPRE} so that the number of large field LPN samples, as used in PPE (Theorem 6.1), is at most a fixed polynomial in the dimension, allowing for the exponent in the sub-exponential indistinguishability to be fixed. (The construction in [JLS22] of PPE relies on large field LPN with dimension k_{PPE} and $k_{\mathsf{PPE}} \cdot n_{\mathsf{PPE}}$ samples, which for us will be k_{CFPRE} and at most $\mathsf{poly}(n_{\mathsf{CFPRE}}, m_{\mathsf{CFPRE}}, k_{\mathsf{CFPRE}}, \lambda)$, respectively.)

Definition 24 (CFPRE Syntax)**.** *A combiner-friendly* CFPRE *scheme, parameterized by some $B = O(1)$, consists of the following p.p.t. algorithms:*

- $\mathsf{PPGen}(1^{n_{\mathsf{CFPRE}}}, 1^{m_{\mathsf{CFPRE}}}, 1^{k_{\mathsf{CFPRE}}}, 1^{\lambda}) \to \mathsf{crs}$. *This algorithm is randomized and outputs a public string $\mathsf{crs} \in \{0,1\}^{\ell_{\mathsf{crs}}}$.*
- $\mathsf{Setup}(1^{n_{\mathsf{CFPRE}}}, 1^{m_{\mathsf{CFPRE}}}, 1^{k_{\mathsf{CFPRE}}}, 1^{\lambda}, \{\mathsf{crs}_j\}_{j\in[B]}) \to \mathsf{MergedCrs}$. *This algorithm deterministically aggregates the information in the individual crs_j's to produce a public reference string* MergedCrs*. (Looking ahead, this will be all of the crs_j's as well as the monomial pattern $\mathcal{Q}_{\{\mathsf{crs}_j\}_{j\in[B]}}$ for us.)*

- $\mathsf{PreProc}(1^{k_{\mathsf{CFPRE}}}, 1^\lambda, n_{\mathsf{CFPRE}}, m_{\mathsf{CFPRE}}, p, \mathbf{x} \in \{0,1\}^{n_{\mathsf{CFPRE}}}, \mathsf{MergedCrs}) \to (\mathsf{PI}, \mathsf{SI})$. *This algorithm is randomized and preprocesses the input* $\mathbf{x}$ *into a string* $(\mathsf{PI}, \mathsf{SI}) \in \mathbb{Z}_p^{\ell_{\mathsf{CFPRE}}}$.
- $\mathsf{Encode}(C, (\mathsf{PI}, \mathsf{SI}), \mathsf{MergedCrs}) \to \mathbf{y}$. *The deterministic encoding algorithm takes as input the circuit* $C \in \mathcal{F}_{\mathsf{CFPRE}, n_{\mathsf{CFPRE}}, m_{\mathsf{CFPRE}}, k_{\mathsf{CFPRE}}, \lambda}$, *preprocessed input* $(\mathsf{PI}, \mathsf{SI})$, *and the* $\mathsf{MergedCrs}$, *and outputs a binary encoding* $\mathbf{y} \in \{0,1\}^*$.
- $\mathsf{Decode}(\mathbf{y}, \mathsf{MergedCrs}) \to \mathsf{out}$. *The deterministic decoding algorithm takes as input an encoding* $\mathbf{y}$ *and outputs a binary output* out.

Definition 25 (CFPRE Correctness). *We say a* CFPRE *scheme satisfies perfect correctness if for all circuits* $C \in \mathcal{F}_{\mathsf{CFPRE}, n_{\mathsf{CFPRE}}, m_{\mathsf{CFPRE}}, k_{\mathsf{CFPRE}}, \lambda}$ *and all* $\mathbf{x} \in \{0,1\}^{n_{\mathsf{CFPRE}}}$, *we have:*

$$\Pr\left[\mathsf{Decode}(\mathbf{y}, \mathsf{MergedCrs}) = C(\mathbf{x}) \;\middle|\; \begin{array}{c} \forall j \in [B] : \mathsf{crs}_j \leftarrow \mathsf{PPGen}(1^{n_{\mathsf{CFPRE}}}, 1^{m_{\mathsf{CFPRE}}}, 1^{k_{\mathsf{CFPRE}}}, 1^\lambda), \\ \mathsf{MergedCrs} \leftarrow \mathsf{Setup}(1^{n_{\mathsf{CFPRE}}}, 1^{m_{\mathsf{CFPRE}}}, 1^{k_{\mathsf{CFPRE}}}, 1^\lambda, \{\mathsf{crs}_j\}_{j \in [B]}), \\ (\mathsf{PI}, \mathsf{SI}) \leftarrow \mathsf{PreProc}(1^{k_{\mathsf{CFPRE}}}, 1^\lambda, n_{\mathsf{CFPRE}}, m_{\mathsf{CFPRE}}, p, \mathbf{x}, \mathsf{MergedCrs}), \\ \mathbf{y} \leftarrow \mathsf{Encode}(C, (\mathsf{PI}, \mathsf{SI}), \mathsf{MergedCrs}) \end{array}\right] = 1.$$

Remark. Definition 6.2 in [JLS22] gives an *imperfect, statistical* correctness definition, where the probability of being correct is $1 - O(\exp(-\lambda^{\Omega(1)}))$ instead of 1. We modify their construction to achieve *perfect* correctness by modifying the underlying use of PPE to achieve the same guarantee (see Remark 6.1).

Definition 26 (CFPRE Security). *We say* CFPRE *is* $\mu(\lambda)$*-secure if there exist distributions*

$$\mathsf{GoodPPGen}(n_{\mathsf{CFPRE}}, m_{\mathsf{CFPRE}}, k_{\mathsf{CFPRE}}, \lambda), \mathsf{BadPPGen}(n_{\mathsf{CFPRE}}, m_{\mathsf{CFPRE}}, k_{\mathsf{CFPRE}}, \lambda)$$

on $\{0,1\}^{\ell_{\mathsf{crs}}}$ *such that both of the following are true:*

1. *For all stateful p.p.t. adversaries* $\mathcal{A}$ *and strings* $\mathsf{good} \in \{0,1\}^B$ *which has 1 in at least one entry, there exists a negligible function* negl *such that* $\mathcal{A}$ *wins the following game with probability* $\leq 1/2 + \mathsf{negl}(\lambda)$:
   - $\mathcal{A}$ *takes as input* 1^λ *and chooses polynomial parameters* $n_{\mathsf{CFPRE}}, m_{\mathsf{CFPRE}}, k_{\mathsf{CFPRE}}$ *and a prime* p *of polynomial length. They then provide two inputs* $\mathbf{x}_0, \mathbf{x}_1 \in \{0,1\}^{n_{\mathsf{CFPRE}}}$ *to the challenger.*
   - *For each* $j \in [B]$, *the challenger samples* $\mathsf{crs}_j \leftarrow \mathsf{GoodPPGen}(n_{\mathsf{CFPRE}}, m_{\mathsf{CFPRE}}, k_{\mathsf{CFPRE}}, \lambda)$ *if* $\mathsf{good}_j = 1$ *and* $\mathsf{BadPPGen}(n_{\mathsf{CFPRE}}, m_{\mathsf{CFPRE}}, k_{\mathsf{CFPRE}}, \lambda)$ *otherwise. They then compute* $\mathsf{MergedCrs} \leftarrow \mathsf{Setup}(1^{n_{\mathsf{CFPRE}}}, 1^{m_{\mathsf{CFPRE}}}, 1^{k_{\mathsf{CFPRE}}}, 1^\lambda, \{\mathsf{crs}_j\}_{j \in [B]})$.
   - *The challenger also samples* $b \leftarrow \{0,1\}$ *and* $(\mathsf{PI}, \mathsf{SI}) \leftarrow \mathsf{PreProc}(1^{k_{\mathsf{CFPRE}}}, 1^\lambda, n_{\mathsf{CFPRE}}, m_{\mathsf{CFPRE}}, p, \mathbf{x}_b, \mathsf{MergedCrs})$. *They send* $\mathsf{PI}, \mathsf{MergedCrs}$ *back to the adversary.*
   - *The adversary replies with a circuit* $C \in \mathcal{F}_{\mathsf{CFPRE}, n_{\mathsf{CFPRE}}, m_{\mathsf{CFPRE}}, k_{\mathsf{CFPRE}}, \lambda}$. *The challenger checks that* $C(\mathbf{x}_0) = C(\mathbf{x}_1)$; *if not, they reject immediately. Otherwise, they sample* $\mathbf{y} \leftarrow \mathsf{Encode}(C, (\mathsf{PI}, \mathsf{SI}), \mathsf{MergedCrs})$ *and send* $\mathbf{y}$ *to the adversary.*
   - *The adversary outputs a guess* $b' \in \{0,1\}$, *and wins if and only if* $b = b'$.

2. *We have $\mu = \Omega(1)$ and the following two distributions are identical:*

   - *Sample* $\mathsf{crs} \leftarrow \mathsf{PPGen}(1^{n_{\mathsf{CFPRE}}}, 1^{m_{\mathsf{CFPRE}}}, 1^{k_{\mathsf{CFPRE}}}, 1^{\lambda})$.
   - *With probability μ, sample* $\mathsf{crs} \leftarrow \mathsf{GoodPPGen}(n_{\mathsf{CFPRE}}, m_{\mathsf{CFPRE}}, k_{\mathsf{CFPRE}}, \lambda)$. *With probability $1-\mu$, sample* $\mathsf{crs} \leftarrow \mathsf{BadPPGen}(n_{\mathsf{CFPRE}}, m_{\mathsf{CFPRE}}, k_{\mathsf{CFPRE}}, \lambda)$.

Furthermore, we say that CFPRE *is sub-exponentially μ-secure if* $\mathsf{negl}(\lambda) = O(\exp(-\lambda^{\Omega(1)}))$.

Remark. We note that this security notion is stronger than that considered by [JLS22], because it allows the adversary to adaptively select the circuit C based on PI and $\{\mathsf{crs}_j\}_{j\in[B]}$. We will require this down the line for compatibility with the FE combiner by [JMS20] and results on bootstrapping CFSKFE to $i\mathcal{O}$ by [KNT22].

Definition 27 (Sublinear Efficiency of CFPRE)**.** *For constants $\tau, c_1, c_2, c_3 > 0$, we say that* CFPRE *satisfies (c_1, c_2, c_3, τ)-sublinear efficiency if the following holds: for all polynomials $n_{\mathsf{CFPRE}}, m_{\mathsf{CFPRE}}, k_{\mathsf{CFPRE}}$, security parameter λ, prime p of length* $\mathsf{poly}(\lambda)$*, and* crs_j*'s in the support of* $\mathsf{PPGen}(1^{n_{\mathsf{CFPRE}}}, 1^{m_{\mathsf{CFPRE}}}, 1^{k_{\mathsf{CFPRE}}}, 1^{\lambda})$*, we have that* $\mathsf{PreProc}(1^{k_{\mathsf{CFPRE}}}, 1^{\lambda}, n_{\mathsf{CFPRE}}, m_{\mathsf{CFPRE}}, p, \cdot, \mathsf{MergedCrs} = \mathsf{Setup}(1^{n_{\mathsf{CFPRE}}}, 1^{m_{\mathsf{CFPRE}}}, 1^{k_{\mathsf{CFPRE}}}, 1^{\lambda}, \{\mathsf{crs}_j\}_{j\in[B]}))$ *is computable by a randomized circuit of size at most*

$$\left((n_{\mathsf{CFPRE}} + m_{\mathsf{CFPRE}}^{1-c_1}) \cdot k_{\mathsf{CFPRE}}^{c_2} + (n_{\mathsf{CFPRE}} + m_{\mathsf{CFPRE}})^{1+\tau} \cdot k_{\mathsf{CFPRE}}^{1-c_3}\right) \cdot \mathsf{poly}(\lambda, \log p).$$

Definition 28 (Polynomial Efficiency of CFPRE)**.** *For any polynomials $n_{\mathsf{CFPRE}}, m_{\mathsf{CFPRE}}, k_{\mathsf{CFPRE}}$, security parameter $\lambda \in \mathbb{N}$, $C \in \mathcal{F}_{\mathsf{CFPRE}, n_{\mathsf{CFPRE}}, m_{\mathsf{CFPRE}}, k_{\mathsf{CFPRE}}, \lambda}$, every prime p, and* crs_j*'s in the support of* $\mathsf{PPGen}(1^{n_{\mathsf{CFPRE}}}, 1^{m_{\mathsf{CFPRE}}}, 1^{k_{\mathsf{CFPRE}}}, 1^{\lambda})$*, define* $\mathsf{MergedCrs} = \mathsf{Setup}(1^{n_{\mathsf{CFPRE}}}, 1^{m_{\mathsf{CFPRE}}}, 1^{k_{\mathsf{CFPRE}}}, 1^{\lambda}, \{\mathsf{crs}_j\}_{j\in[B]})$. *Then we say that* CFPRE *has degree d encoding if there exists a polynomial f over $\mathbb{Z}_p$ satisfying the following:*

- *For every input $\mathbf{x} \in \{0,1\}^{n_{\mathsf{CFPRE}}}$, and every* $(\mathsf{PI}, \mathsf{SI})$ *in the support of* $\mathsf{PreProc}(1^{k_{\mathsf{CFPRE}}}, 1^{\lambda}, n_{\mathsf{CFPRE}}, m_{\mathsf{CFPRE}}, p, \mathbf{x}, \mathsf{crs})$*, we have*

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

- *f has degree $\leq d$ in* PI *and degree 2 in* SI.

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

In this section, we prove the following theorem:

Theorem 6.2. *Suppose for all $B = O(1)$, there exist constants $d \in \mathbb{N}$ and $c_1, c_2, c_3 > 0$ such that for all $\tau > 0$, there exists a (sub-exponentially) μ-secure c_3-sublinear and (d, c_1, c_2, τ)-efficient* CFARE*. Additionally, assume there there exists a (sub-exponentially) secure* PPE *scheme* $\mathcal{F}_{\mathsf{PPE},d}$.

Then for all $B = O(1)$, there exist constants $d' \in \mathbb{N}, c'_1, c'_2, c'_3 > 0$ such that for all $\tau > 0$, there exists a (sub-exponentially) μ-secure (c'_1, c'_2, c'_3, τ)-sublinear CFPRE *scheme with degree d' encoding.*

By applying Theorems 6.1 and 5.1, we directly obtain the following corollary:

Corollary 6.3. *Suppose that the (sub-exponential) large-field* LPN *and* SparseLPN *assumptions hold. Then for all $B = O(1)$, there exist constants $d \in \mathbb{N}$ and $c_1, c_2, c_3 > 0$ such that for all $\tau > 0$, there exists a (sub-exponentially) $\Omega(1)$-secure (c_1, c_2, c_3, τ)-sublinear* CFPRE *scheme with degree d encoding.*

6.3 Construction Details

Our construction closely mirrors that of [JLS22], and simply composes PPE and CFARE. We specify ingredients and parameters below. Let $d \in \mathbb{N}$ and $c_1, c_2 > 0$ be efficiency parameters inherited from the CFARE as specified in Theorem 5.1.

1. For the CFARE scheme, we set:
   - $n_{\mathsf{CFARE}} = n_{\mathsf{CFPRE}}$,
   - $m_{\mathsf{CFARE}} = m_{\mathsf{CFPRE}}$,
   - $k_{\mathsf{CFARE}} = k_{\mathsf{CFPRE}}$ (note that these settings imply that $\mathcal{F}_{\mathsf{CFPRE}, n_{\mathsf{CFPRE}}, m_{\mathsf{CFPRE}}, k_{\mathsf{CFPRE}}, \lambda} = \mathcal{F}_{\mathsf{CFARE}, n_{\mathsf{CFARE}}, m_{\mathsf{CFARE}}, k_{\mathsf{CFARE}}, \lambda}$),
   - $m'_{\mathsf{CFARE}} = O((n_{\mathsf{CFPRE}} + m_{\mathsf{CFPRE}})^{1+\tau}\lambda^{c_1})$ (which is an upper bound on the number of monomials in $\mathcal{Q}_{\{\mathsf{crs}_j\}_{j\in[B]}}$, where the crs_j's will be sampled from $\mathsf{CFARE.Setup}$), and
   - $n'_{\mathsf{CFARE}} = O((n_{\mathsf{CFPRE}} + m_{\mathsf{CFPRE}}^{1-c_2})\lambda^{c_1})$ (which is the length of the inputs $\mathbf{a}_i$ to the polynomial computing $\mathsf{CFARE.Encode}$).
2. For the PPE scheme, we set:
   - The prime to be used as p,
   - $n_{\mathsf{PPE}} = n'_{\mathsf{CFARE}}$,
   - $m_{\mathsf{PPE}} = m'_{\mathsf{CFARE}}$,
   - $k_{\mathsf{PPE}} = k_{\mathsf{CFARE}}$,
   - Let $d' = O(d)$ be an upper bound on the constant degree on the multivariate polynomial $g_{f,\mathcal{Q}}(\cdot,\cdot) = \mathsf{PPE.Eval}(f, (\cdot,\cdot)) \bmod p$ over $\mathbb{Z}_p$, for any d-monomial pattern $\mathcal{Q}$ over n_{PPE} variables with $|\mathcal{Q}| \leq m_{\mathsf{PPE}}$ and $f \in \mathcal{F}_{\mathsf{PPE}, d, n_{\mathsf{PPE}}, \mathcal{Q}, k_{\mathsf{PPE}}}$.

Our construction is described in Figure 6. We now argue that our construction satisfies all the necessary properties:

Correctness: By correctness of PPE, the output $\mathbf{y}$ of $\mathsf{CFPRE.Encode}(C, (\mathsf{PI}, \mathsf{SI}), \mathsf{MergedCrs})$ will be exactly $(f_1(\mathsf{PI}, \mathsf{SI}), \ldots, f_T(\mathsf{PI}, \mathsf{SI})) = \mathsf{CFARE.Encode}(C, \mathbf{x}, \{\mathsf{CFARE.crs}_j\}_{j\in[B]}, \mathbf{r})$. Then by correctness of CFARE, it will follow that $\mathsf{CFARE.Decode}$ (and hence $\mathsf{CFPRE.Decode}$) outputs $C(\mathbf{x})$.

Security: We will take $\mathsf{GoodPPGen}(1^{n_{\mathsf{CFPRE}}}, 1^{m_{\mathsf{CFPRE}}}, 1^{k_{\mathsf{CFPRE}}}, \lambda)$ to sample from

$$\mathsf{CFARE.GoodSetup}(1^{n_{\mathsf{CFARE}}}, 1^{m_{\mathsf{CFARE}}}, 1^{k_{\mathsf{CFARE}}}, \lambda),$$

and likewise $\mathsf{BadPPGen}$ will sample from $\mathsf{CFARE.BadSetup}$. With this in mind, we prove security for this choice of $\mathsf{GoodPPGen}$ and $\mathsf{BadPPGen}$. We will provide a series of hybrid security games for a fixed string $\mathsf{good} \in \{0,1\}^B$ with at least one entry being 1, starting from the CFPRE security game defined in Definition 26, and ending with a game that does not depend at all on the challenger's bit b.

Hybrid_0: This will be exactly the security game in Definition 26.

CFPRE Construction

$\mathsf{CFPRE.PPGen}(1^{n_{\mathsf{CFPRE}}}, 1^{m_{\mathsf{CFPRE}}}, 1^{k_{\mathsf{CFPRE}}}, 1^{\lambda})$:

1. Compute and output $\mathsf{CFARE.crs} \leftarrow \mathsf{CFARE.Setup}(1^{n_{\mathsf{CFARE}}}, 1^{m_{\mathsf{CFARE}}}, 1^{k_{\mathsf{CFARE}}}, 1^{\lambda})$.

$\mathsf{CFPRE.Setup}(1^{n_{\mathsf{CFPRE}}}, 1^{m_{\mathsf{CFPRE}}}, 1^{k_{\mathsf{CFPRE}}}, \{\mathsf{CFARE.crs}_j\}_{j\in[B]})$:

1. Compute the monomial pattern $\mathcal{Q}_{\{\mathsf{CFARE.crs}_j\}_{j\in[B]}}$ and output $\mathsf{MergedCrs} = (\{\mathsf{CFARE.crs}_j\}_{j\in[B]}, \mathcal{Q}_{\{\mathsf{CFARE.crs}_j\}_{j\in[B]}})$.

$\mathsf{CFPRE.PreProc}(1^{k_{\mathsf{CFPRE}}}, 1^{\lambda}, n_{\mathsf{CFPRE}}, m_{\mathsf{CFPRE}}, p, \mathbf{x}, \mathsf{MergedCrs})$:

1. Compute $\mathbf{r} \leftarrow \mathsf{CFARE.SdSamp}(1^{k_{\mathsf{CFARE}}}, 1^{\lambda}, n_{\mathsf{CFARE}}, m_{\mathsf{CFARE}})$, and parse it as $\{\mathbf{r}_{i,j}\}_{i\in[k_{\mathsf{CFARE}}], j\in[B]}$.
2. For each $i \in [k_{\mathsf{CFARE}}]$, set $\mathbf{a}_i = (\mathbf{x}, \mathbf{r}_i = \{\mathbf{r}_{i,j}\}_{j\in[B]}) \in \{0,1\}^{n'_{\mathsf{CFARE}}=n_{\mathsf{PPE}}}$.
3. Compute and output
$$(\mathsf{PI}, \mathsf{SI}) \leftarrow \mathsf{PPE.PreProc}(1^{n_{\mathsf{PPE}}}, 1^{k_{\mathsf{PPE}}}, p, \mathcal{Q}_{\{\mathsf{CFARE.crs}_j\}_{j\in[B]}}, (\mathbf{a}_1, \ldots, \mathbf{a}_{k_{\mathsf{CFPRE}}})).$$
Note that the final argument here has length $k_{\mathsf{CFPRE}} \cdot n'_{\mathsf{CFARE}} = k_{\mathsf{PPE}} \cdot n_{\mathsf{PPE}}$, so this conforms to the PPE syntax.

$\mathsf{CFPRE.Encode}(C, (\mathsf{PI}, \mathsf{SI}), \mathsf{MergedCrs} = (\{\mathsf{CFARE.crs}_j\}_{j\in[B]}, \mathcal{Q}_{\{\mathsf{CFARE.crs}_j\}_{j\in[B]}}))$:

1. Let T be the output length of CFARE. Then, by the polynomial efficiency property of CFARE, for any circuit $C \in \mathcal{F}_{\mathsf{CFPRE}, n_{\mathsf{CFPRE}}, m_{\mathsf{CFPRE}}, k_{\mathsf{CFPRE}}, \lambda} = \mathcal{F}_{\mathsf{CFARE}, n_{\mathsf{CFARE}}, m_{\mathsf{CFARE}}, k_{\mathsf{CFARE}}, \lambda}$ and $i \in [T]$, the ith output bit of $\mathsf{CFARE.Encode}(C, \cdot, \{\mathsf{CFARE.crs}_j\}_{j\in[B]}, \cdot)$ is computable by an efficiently generatable polynomial $f_i \in \mathcal{F}_{\mathsf{PPE}, d, n_{\mathsf{PPE}}, \mathcal{Q}_{\{\mathsf{CFARE.crs}_j\}_{j\in[B]}}, k_{\mathsf{PPE}}}$. Let g_{f_i} be the degree $(d', 2)$-polynomial evaluating $\mathsf{PPE.Eval}(f_i, \cdot)$.
2. Compute $y_i = \mathsf{PPE.Eval}(f_i, \mathsf{PI}, \mathsf{SI}) = g_{f_i}(\mathsf{PI}, \mathsf{SI})$ over $\mathbb{Z}_p$.
3. Output $\mathbf{y} = (y_1, \ldots, y_T)$.

$\mathsf{CFPRE.Decode}(\mathbf{y}, \mathsf{MergedCrs} = (\{\mathsf{CFARE.crs}_j\}_{j\in[B]}, \mathcal{Q}_{\{\mathsf{CFARE.crs}_j\}_{j\in[B]}}))$:

1. Compute and output $\mathsf{CFARE.Decode}(1^{n_{\mathsf{CFARE}}}, 1^{m_{\mathsf{CFARE}}}, 1^{k_{\mathsf{CFARE}}}, 1^{\lambda}, \mathbf{y}, \{\mathsf{CFARE.crs}_j\}_{j\in[B]})$.

Figure 6: Our CFPRE Construction.

Hybrid_1: Instead of computing $y_i \leftarrow \mathsf{PPE.Eval}(f_i, (\mathsf{PI}, \mathsf{SI}))$ for each $i \in [T]$, the challenger will simply compute $\mathbf{y} \leftarrow \mathsf{CFARE.Encode}(C, \mathbf{x}_b, \{\mathsf{CFARE.crs}_j\}_{j\in[B]}, \mathbf{r})$ and sends this to the adversary.

This is identical to Hybrid_0 by correctness of the PPE scheme.

Hybrid_2: Instead of computing $(\mathsf{PI}, \mathsf{SI}) \leftarrow \mathsf{PPE.PreProc}(1^{n_{\mathsf{PPE}}}, 1^{k_{\mathsf{PPE}}}, p, \mathcal{Q}_{\{\mathsf{CFARE.crs}_j\}_{j\in[B]}}, (\mathbf{a}_1, \dots, \mathbf{a}_{k_{\mathsf{CFPRE}}}))$, the challenger will preprocess the all 0's string, i.e., they will compute

$$(\mathsf{PI}, \mathsf{SI}) \leftarrow \mathsf{PPE.PreProc}(1^{n_{\mathsf{PPE}}}, 1^{k_{\mathsf{PPE}}}, p, \mathcal{Q}_{\{\mathsf{CFARE.crs}_j\}_{j\in[B]}}, \mathbf{0}).$$

(As before, $\mathbf{y}$ will still be computed as in Hybrid_1.)

Since the adversary only sees PI, this is computationally indistinguishable from Hybrid_1 by PPE security. The security loss will be $\mathsf{negl}(k_{\mathsf{PPE}}) \leq \mathsf{negl}(\lambda)$. (We note as in Remark 5.2 that this holds even though $\mathsf{GoodPPGen}$ and $\mathsf{BadPPGen}$ may not be efficient.)

Hybrid_3: $(\mathsf{PI}, \mathsf{SI})$ will be computed as in Hybrid_2, but now the challenger will compute

$$\mathbf{y} \leftarrow \mathsf{CFARE.Encode}(C, \mathbf{x}_0, \{\mathsf{CFARE.crs}_j\}_{j\in[B]}, \mathbf{r}).$$

In both games, the adversary will see $\mathbf{y}$ along with $\mathsf{MergedCrs} = (\{\mathsf{CFARE.crs}_j\}_{j\in[B]}, \mathcal{Q}_{\{\mathsf{CFARE.crs}_j\}_{j\in[B]}})$ and PI.

Since $\mathcal{Q}_{\{\mathsf{CFARE.crs}_j\}_{j\in[B]}}$ can be efficiently computed from $\{\mathsf{CFARE.crs}_j\}_{j\in[B]}$, this is computationally indistinguishable from Hybrid_2 by indistinguishability security of the CFARE (noting that our CFARE security game allows the adversary to select C adaptively based on $\mathsf{MergedCrs}$). Here, we are using the fact that $\mathsf{good} \in \{0,1\}^B$ contains a 1 in at least one entry, as that is needed to apply CFARE security.

The adversary's view in this hybrid is now completely independent of b, as desired.

Sublinear Efficiency: We analyze the runtime of each part of $\mathsf{CFPRE.PreProc}$ individually. By sublinear efficiency of the CFARE, the call to $\mathsf{CFARE.SdSamp}$ is computable by a randomized circuit of size at most:

$$O(k_{\mathsf{CFARE}} \cdot (n_{\mathsf{CFARE}} + m_{\mathsf{CFARE}}^{1-c_3}) \cdot \mathsf{poly}(\lambda)) = k_{\mathsf{CFPRE}} \cdot (n_{\mathsf{CFPRE}} + m_{\mathsf{CFPRE}}^{1-c_3}) \cdot \mathsf{poly}(\lambda).$$

Parsing $\mathbf{a}_1, \dots, \mathbf{a}_{k_{\mathsf{CFARE}}}$ is trivially doable with a circuit of size:

$$O(k_{\mathsf{CFARE}} \cdot n'_{\mathsf{CFARE}}) = (k_{\mathsf{CFPRE}} \cdot (n_{\mathsf{CFPRE}} + m_{\mathsf{CFPRE}}^{1-c_2})) \cdot \mathsf{poly}(\lambda).$$

Finally, by the sublinear efficiency of the PPE, the call to $\mathsf{PPE.PreProc}$ is computable with a randomized circuit of size at most $(n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}^{c_4} + m_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}^{1-c_5}) \cdot \mathsf{poly}(\log p)$ for some constants $c_4, c_5 > 0$. We break down each of these terms separately:

$$\begin{aligned} n_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}^{c_4} \cdot \mathsf{poly}(\log p) &= n'_{\mathsf{CFARE}} \cdot k_{\mathsf{PPE}}^{c_4} \cdot \mathsf{poly}(\log p) \\ &= (n_{\mathsf{CFPRE}} + m_{\mathsf{CFPRE}}^{1-c_2}) \cdot k_{\mathsf{CFPRE}}^{c_4} \cdot \mathsf{poly}(\lambda, \log p), \text{ and} \\ m_{\mathsf{PPE}} \cdot k_{\mathsf{PPE}}^{1-c_5} \cdot \mathsf{poly}(\log p) &= m'_{\mathsf{CFARE}} \cdot k_{\mathsf{PPE}}^{1-c_5} \cdot \mathsf{poly}(\log p) \\ &= (n_{\mathsf{CFPRE}} + m_{\mathsf{CFPRE}})^{1+\tau} \cdot k_{\mathsf{CFPRE}}^{1-c_5} \cdot \mathsf{poly}(\lambda, \log p). \end{aligned}$$

Putting everything together, the overall size of the circuit is at most:

$$\left((n_{\mathsf{CFPRE}} + m_{\mathsf{CFPRE}}^{1-\min(c_2,c_3)}) \cdot k_{\mathsf{CFPRE}}^{\max(1,c_4)} + (n_{\mathsf{CFPRE}} + m_{\mathsf{CFPRE}})^{1+\tau} \cdot k_{\mathsf{CFPRE}}^{1-c_5}\right) \cdot \mathsf{poly}(\lambda, \log p).$$

Hence our CFPRE satisfies $(\min(c_2, c_3), \max(1, c_4), c_5, \tau)$-sublinear efficiency.

Polynomial Efficiency: Note that $\mathsf{CFPRE.Encode}(C, (\mathsf{PI}, \mathsf{SI}), \mathsf{MergedCrs})$ computes $\mathbf{y} = (g_{f_1}(\mathsf{PI}, \mathsf{SI}), \ldots, g_{f_T}(\mathsf{PI}, \mathsf{SI}))$, where $f_i(\mathbf{a})$ computes the ith bit of $\mathsf{CFARE.Encode}(C, \mathbf{x}, \{\mathsf{CFARE.crs}_j\}_{j\in[B]}, \mathbf{r})$. From the polynomial efficiency of the CFARE, f_i is of the form

$$f_i(\mathbf{a}) = \sum_{Q \in \mathcal{Q}_{\{\mathsf{CFARE.crs}_i\}_{i\in[B]}}, j\in[k_{\mathsf{CFARE}}]} \mu_{i,Q,j} \cdot \mathsf{Mon}_Q(\mathbf{a}_j),$$

where $\mu_{i,Q,j} \in \mathbb{Z}$. Hence $f_i \in \mathcal{F}_{\mathsf{PPE},d,n_{\mathsf{PPE}},\mathcal{Q}_{\{\mathsf{CFARE.crs}_j\}_{j\in[B]}},k_{\mathsf{PPE}}}$. By the complexity requirement of PPE, g_{f_i} will be a polynomial over $\mathbb{Z}_p$ of degree $(d', 2)$. Hence the polynomial efficiency of CFPRE with degree $d' = O(d)$ encoding is satisfied by the polynomial $(g_{f_1}, g_{f_2}, \ldots, g_{f_T})$, i.e., this polynomial has outputs in $\mathbb{Z}_p^T$.

In summary, we have shown the existence of constants $c_1' = \min(c_2, c_3)$, $c_2' = \max(1, c_4)$, $c_3' = c_5$, and $d' = O(d) \in \mathbb{N}$ such that for any $\tau > 0$, there exists a (c_1', c_2', c_3', τ)-sublinear CFPRE scheme with degree d' encoding. This completes our proof of Theorem 6.2.

7 Functional Encryption

Let $\mathcal{F}_{\mathsf{CFPKFE}} = \{\mathcal{F}_{\mathsf{CFPKFE},n_{\mathsf{FE}},m_{\mathsf{FE}},\lambda}\}_{n_{\mathsf{FE}}(\cdot),m_{\mathsf{FE}}(\cdot),\lambda\in\mathbb{N}}$ be the class of all boolean circuits with $n_{\mathsf{FE}}(\lambda)$ input bits and $m_{\mathsf{FE}}(\lambda)$ output bits, where every output bit can be computed by a circuit of size λ. Also, let $\mathcal{F}_{\mathsf{FE}} = \{\mathcal{F}_{\mathsf{FE},n_{\mathsf{FE}},s_{\mathsf{FE}},\lambda}\}_{n_{\mathsf{FE}}(\cdot),s_{\mathsf{FE}}(\cdot),\lambda\in\mathbb{N}}$ be the class of all boolean circuits with $n_{\mathsf{FE}}(\lambda)$ input bits and size at most $s_{\mathsf{FE}}(\lambda)$, without a constraint on the number of output bits. Here, n_{FE}, m_{FE}, and s_{FE} are all polynomials in λ, where we assume that $\lambda \le m_{\mathsf{FE}}$ and $n_{\mathsf{FE}} \le m_{\mathsf{FE}}$. This will turn out to be without loss of generality; jumping ahead, our sublinear efficiency property (Definition 36) can tolerate any fixed polynomial dependence on n_{FE} and λ.

Remark. The reason we assume $\lambda \le m_{\mathsf{FE}}$ is to ensure that k_{CFPRE} is lower bounded by a fixed polynomial in λ (as needed for CFPRE); in the construction of CFPKFE, we will set k_{CFPRE} to be a fixed polynomial in m_{FE}. Similarly, we assume $n_{\mathsf{FE}} \le m_{\mathsf{FE}}$ to ensure that n_{FE} is also upper-bounded by a fixed polynomial in k_{CFPRE} (as needed for CFPRE); in our construction of CFPKFE, we will set n_{CFPRE} to be n_{FE}.

7.1 Ingredient: Partially Hiding Functional Encryption

Here, we define partially hiding functional encryption as constructed by [Wee20], which will be our main tool to bootstrap CFPRE to FE. The input to such a scheme has a public part PI and a secret part SI. The decryptor with the function key for a particular f should only be able to learn PI and SI.

The construction of [Wee20] shows that this can be realized for any function over $\mathbb{Z}_p$ that is degree $O(1)$ in PI and degree at most 2 in SI. Formally, we define the function class $\mathcal{F}_{\mathsf{PHFE}} =$

$\{\mathcal{F}_{\mathsf{PHFE},d,p,n_{\mathsf{PHFE}}}\}_{d\in\mathbb{N},p \text{ prime},n_{\mathsf{PHFE}}\in\mathbb{N}}$ to comprise all polynomials f that take as input $\mathsf{PI},\mathsf{SI}\in\mathbb{Z}_p^{n_{\mathsf{PHFE}}}$ and has the form

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

where each $f_{j,k}$ has degree at most d.

Definition 29 (PHFE Syntax). *A public key partially hiding functional encryption scheme for $\mathcal{F}_{\mathsf{PHFE}}$ consists of the following polynomial time algorithms:*

- $\mathsf{PPGen}(1^\lambda)$*: a randomized algorithm that takes as input the security parameter λ and outputs a string $\mathsf{PP}=(\mathsf{crs},p)$ which includes the prime modulus p.*
- $\mathsf{Setup}(d,1^{n_{\mathsf{PHFE}}},\mathsf{PP})$*: a randomized algorithm that outputs a public key PK and master secret key MSK.*
- $\mathsf{Enc}(\mathsf{PK},(\mathsf{PI},\mathsf{SI})\in\mathbb{Z}_p^{n_{\mathsf{PHFE}}}\times\mathbb{Z}_p^{n_{\mathsf{PHFE}}})$*: a randomized algorithm that outputs a ciphertext CT. We implicitly assume that CT includes PK in the clear.*
- $\mathsf{KeyGen}(\mathsf{MSK},f\in\mathcal{F}_{\mathsf{PHFE},d,p,n_{\mathsf{PHFE}}})$*: a randomized algorithm that takes as input a degree $(d,2)$-polynomial f over $\mathbb{Z}_p$ and outputs a decryption key SK_f for f.*
- $\mathsf{Dec}(\mathsf{SK}_f,\mathsf{CT})$*: a deterministic algorithm that returns a value out, which is either $\perp$ or an integer.*

Definition 30 (PHFE Correctness). *A PHFE scheme PHFE is correct if for any $d,\lambda\in\mathbb{N}$, polynomial $n_{\mathsf{PHFE}}=n_{\mathsf{PHFE}}(\lambda)$, $(\mathsf{crs},p)\leftarrow\mathsf{PPGen}(1^\lambda)$, $(\mathsf{PI},\mathsf{SI})\in\mathbb{Z}_p^{n_{\mathsf{PHFE}}}\times\mathbb{Z}_p^{n_{\mathsf{PHFE}}}$, and every function $f\in\mathcal{F}_{\mathsf{PHFE},d,p,n_{\mathsf{PHFE}}}$ such that $f(\mathsf{PI},\mathsf{SI})\in\{0,1\}$, we have:*

$$\Pr\left[\mathsf{Dec}(\mathsf{SK}_f,\mathsf{CT})=f(\mathsf{PI},\mathsf{SI})\;\middle|\;\begin{array}{c}(\mathsf{PK},\mathsf{MSK})\leftarrow\mathsf{Setup}(d,1^{n_{\mathsf{PHFE}}},\mathsf{PP}),\\ \mathsf{CT}\leftarrow\mathsf{Enc}(\mathsf{PK},(\mathsf{PI},\mathsf{SI})),\\ \mathsf{SK}_f\leftarrow\mathsf{KeyGen}(\mathsf{SK},f)\end{array}\right]=1.$$

Definition 31 (PHFE Simulation Security). *We say PHFE for functionality $\mathcal{F}_{\mathsf{PHFE}}$ is (selective) SIM secure, if there additionally exist p.p.t. algorithms $\widetilde{\mathsf{Setup}},\widetilde{\mathsf{Enc}},\widetilde{\mathsf{KeyGen}}$ with the following property. For every constant $d>0$ and every polynomial $n_{\mathsf{PHFE}}=n_{\mathsf{PHFE}}(\lambda)$ and number of queries $Q_{\mathsf{SK}}=Q_{\mathsf{SK}}(\lambda)$, with probability $1-\mathsf{negl}_1(\lambda)$ over the randomness of $(\mathsf{crs},p)=\mathsf{PP}\leftarrow\mathsf{PPGen}(1^\lambda)$, the following two games are computationally indistinguishable by any stateful p.p.t. adversary $\mathcal{A}$ with advantage bounded by $\mathsf{negl}_2(\lambda)$:*

1. *Game 1:*
   - *The challenger samples $(\mathsf{PK},\mathsf{MSK})\leftarrow\mathsf{Setup}(d,1^{n_{\mathsf{PHFE}}},\mathsf{PP})$ and sends PK to $\mathcal{A}$.*
   - *The adversary responds with a query $(\mathsf{PI},\mathsf{SI})$.*
   - *The challenger replies with $\mathsf{Enc}(\mathsf{PK},(\mathsf{PI},\mathsf{SI}))$.*
   - *The adversary can then adaptively make at most $Q_{\mathsf{SK}}(\lambda)$ queries of the following form: they send the challenger a function $f\in\mathcal{F}_{\mathsf{PHFE},d,p,n_{\mathsf{PHFE}}}$ and the challenger replies with $\mathsf{SK}_f\leftarrow\mathsf{KeyGen}(\mathsf{MSK},f)$.*

2. *Game 2:*

   - *The challenger samples* $(\widetilde{\mathsf{PK}}, \widetilde{\mathsf{MSK}}) \leftarrow \widetilde{\mathsf{Setup}}(d, 1^{n_{\mathsf{PHFE}}}, \mathsf{PP})$ *and sends* $\widetilde{\mathsf{PK}}$ *to* $\mathcal{A}$.
   - *The adversary constructs a query* $(\mathsf{PI}, \mathsf{SI})$ *and sends* PI *to the challenger.*
   - *The challenger replies with* $\widetilde{\mathsf{Enc}}(\widetilde{\mathsf{MSK}}, \mathsf{PI})$.
   - *The adversary can then adaptively make at most* $Q_{\mathsf{SK}}(\lambda)$ *queries of the following form: they send the challenger a function* $f \in \mathcal{F}_{\mathsf{PHFE},d,p,n_{\mathsf{PHFE}}}$ *and the challenger replies with* $\widetilde{\mathsf{SK}}_f \leftarrow \widetilde{\mathsf{KeyGen}}(\widetilde{\mathsf{MSK}}, \mathsf{PI}, f, f(\mathsf{PI}, \mathsf{SI}))$.

Moreover, we say the scheme satisfies sub-exponential SIM security if $\mathsf{negl}_1(\lambda)$ *and* $\mathsf{negl}_2(\lambda)$ *are* $O(\exp(-\lambda^{\Omega(1)}))$.

Remark. This definition is stronger than the definition used by [JLS22], as it allows the adversary some level of adaptivity (called semi-adaptivity [CW14]), where the choice of the function f can depend on the challenge ciphertext. However, the original construction of Wee [Wee20] works with this stronger definition, and we will need it later in Section 7.4 when bootstrapping our eventual FE construction to $i\mathcal{O}$.

The reason for this difference in definitions is that [JLS22] uses the bootstrapping results of [BV18, AJ15] from PKFE to $i\mathcal{O}$, which only require security against completely non-adaptive adversaries. On the other hand, the bootstrapping results by [KNT17, KNT22] from SKFE to $i\mathcal{O}$ that we use require security against semi-adaptive adversaries.

Definition 32 (PHFE Linear Efficiency)**.** *We say that* PHFE *satisfies linear efficiency if for any constant* $d \in \mathbb{N}$, *there exists a fixed polynomial* poly *(that might depend on* d*) so that: for any polynomial* $n_{\mathsf{PHFE}}(\lambda)$, *for any* $(\mathsf{crs}, \mathsf{PP})$ *in the support of* $\mathsf{PPGen}(1^\lambda)$ *and* $(\mathsf{PK}, \mathsf{MSK})$ *in the support of* $\mathsf{Setup}(d, 1^{n_{\mathsf{PHFE}}(\lambda)}, \mathsf{PP})$, *the size of the circuit* $\mathsf{Enc}(\mathsf{PK}, (\cdot, \cdot))$ *is* $O(n_{\mathsf{PHFE}}(\lambda) \cdot \mathsf{poly}(\lambda))$. *Moreover, this circuit is uniformly and efficiently generatable.*

We have the following theorem:

Theorem 7.1 ([JLMS19, Wee20, GJLS21])**.** *If the* DLIN *assumption over prime order symmetric bilinear groups holds, there exists a* PHFE *scheme. Moreover, if the assumption is sub-exponentially secure, the resulting* PHFE *scheme is also sub-exponentially secure.*

7.2 Public-Key Functional Encryption

In this section, we follow the construction of [JLS22] to construct public-key functional encryption from CFPRE and the PHFE described in Section 7.1. We define CFPKFE here for the function family $\mathcal{F}_{\mathsf{CFPKFE}}$, but it can also be defined analogously for $\mathcal{F}_{\mathsf{FE}}$.

Definition 33 (CFPKFE Syntax)**.** *A combiner-friendly public-key functional encryption scheme* CFPKFE, *parameterized by some* $B = O(1)$, *for the function class* $\mathcal{F}_{\mathsf{CFPKFE},n_{\mathsf{FE}},m_{\mathsf{FE}},\lambda}$ *consists of the following p.p.t. algorithms:*

- $\mathsf{PPGen}(1^\lambda, 1^{n_{\mathsf{FE}}}, m_{\mathsf{FE}})$: *this is a randomized algorithm that samples a* crs.

- $\mathsf{Setup}(1^\lambda, 1^{n_{\mathsf{FE}}}, m_{\mathsf{FE}}, \{\mathsf{crs}_j\}_{j\in[B]})$*: this is a randomized algorithm that outputs a public key* PK*, master secret key* MSK*, and* $\mathsf{MergedCrs}$*. For us, B will be a universal constant, and the* crs_i*'s will be independent samples from* PPGen*.*
- $\mathsf{Enc}(\mathsf{PK}, \mathbf{x} \in \{0,1\}^{n_{\mathsf{FE}}}, \mathsf{MergedCrs})$*: this is randomized algorithm that outputs a ciphertext* CT*.*
- $\mathsf{KeyGen}(\mathsf{MSK}, f \in \mathcal{F}_{\mathsf{CFPKFE},n_{\mathsf{FE}},m_{\mathsf{FE}},\lambda}, \mathsf{MergedCrs})$*: this randomized algorithm outputs a functional decryption key* SK_f*.*
- $\mathsf{Dec}(\mathsf{SK}_f, \mathsf{CT}, \mathsf{MergedCrs})$*: deterministically outputs a value $\mathbf{y} \in \{0,1\}^{m_{\mathsf{FE}}}$, or $\perp$ if it fails.*

Definition 34 (CFPKFE Correctness)**.** *We require that for any polynomials $n_{\mathsf{FE}}(\lambda)$, $m_{\mathsf{FE}}(\lambda)$, any function $f \in \mathcal{F}_{\mathsf{CFPKFE},n_{\mathsf{FE}},m_{\mathsf{FE}},\lambda}$, and any $\mathbf{x} \in \{0,1\}^{n_{\mathsf{FE}}}$, we have:*

$$\Pr\left[\mathsf{Dec}(\mathsf{SK}_f, \mathsf{CT}, \mathsf{MergedCrs}) = f(\mathbf{x}) \;\middle|\; \begin{array}{c} \forall i \in [B], \mathsf{crs}_i \leftarrow \mathsf{PPGen}(1^\lambda, 1^{n_{\mathsf{FE}}}, m_{\mathsf{FE}}) \\ (\mathsf{PK}, \mathsf{MSK}, \mathsf{MergedCrs}) \leftarrow \mathsf{Setup}(1^\lambda, 1^{n_{\mathsf{FE}}}, m_{\mathsf{FE}}, \{\mathsf{crs}_i\}_{i\in[B]}), \\ \mathsf{CT} \leftarrow \mathsf{Enc}(\mathsf{PK}, \mathbf{x}, \mathsf{MergedCrs}), \\ \mathsf{SK}_f \leftarrow \mathsf{KeyGen}(\mathsf{MSK}, f, \mathsf{MergedCrs}) \end{array}\right] = 1.$$

Definition 35 (CFPKFE Security)**.** *We say* CFPKFE *is (selective) IND μ-secure if there exist (possibly inefficient) algorithms* $\mathsf{GoodPPGen}$ *and* $\mathsf{BadPPGen}$ *with the same syntax as* PPGen *such that both of the following hold:*

1. *For all stateful p.p.t. adversaries $\mathcal{A}$ and binary strings* $\mathsf{good} \in \{0,1\}^B$ *such that at least one entry of* good *is 1, there exists a negligible function* negl *such that $\mathcal{A}$ succeeds in the below game with probability at most $1/2 + \mathsf{negl}(\lambda)$:*
   - *$\mathcal{A}$ takes as input 1^λ and chooses input and output lengths n_{FE} and m_{FE}. They then provide two inputs $\mathbf{x}_0, \mathbf{x}_1 \in \{0,1\}^{n_{\mathsf{FE}}}$ to the challenger.*
   - *For each $i \in [r]$, the challenger samples $\mathsf{crs}_i \leftarrow \mathsf{GoodPPGen}(1^\lambda, 1^{n_{\mathsf{FE}}}, m_{\mathsf{FE}})$ if $\mathsf{good}_i = 1$ and $\mathsf{BadPPGen}(1^\lambda, 1^{n_{\mathsf{FE}}}, m_{\mathsf{FE}})$ otherwise. They then sample $(\mathsf{PK}, \mathsf{MSK}, \mathsf{MergedCrs}) \leftarrow \mathsf{Setup}(1^\lambda, 1^{n_{\mathsf{FE}}}, m_{\mathsf{FE}}, \{\mathsf{crs}_i\}_{i\in[B]})$.*
   - *The challenger now samples $b \leftarrow \{0,1\}$. They send $\mathcal{A}$ the public key* PK*,* $\mathsf{MergedCrs}$*, and the ciphertext $\mathsf{CT} \leftarrow \mathsf{Enc}(\mathsf{PK}, \mathbf{x}_b, \mathsf{MergedCrs})$.*
   - *$\mathcal{A}$ then sends a function $f \in \mathcal{F}_{\mathsf{CFPKFE},n_{\mathsf{FE}},m_{\mathsf{FE}},\lambda}$. The challenger checks that $f(\mathbf{x}_0) = f(\mathbf{x}_1)$. If not, they reject immediately.*
   - *The challenger sends the function key $\mathsf{SK}_f \leftarrow \mathsf{KeyGen}(\mathsf{MSK}, f, \mathsf{MergedCrs})$ to the adversary.*
   - *The adversary outputs a guess $b' \in \{0,1\}$ and wins if and only if $b = b'$ (and the challenger did not reject already).*
2. *There exists $\mu(\lambda) = \Omega(1)$ such that the following two distributions are identical:*
   - *Sample $\mathsf{crs} \leftarrow \mathsf{PPGen}(1^\lambda, 1^{n_{\mathsf{FE}}}, m_{\mathsf{FE}})$.*
   - *With probability μ, sample $\mathsf{crs} \leftarrow \mathsf{GoodPPGen}(1^\lambda, 1^{n_{\mathsf{FE}}}, m_{\mathsf{FE}})$. With probability $1-\mu$, sample $\mathsf{crs} \leftarrow \mathsf{BadPPGen}(1^\lambda, 1^{n_{\mathsf{FE}}}, m_{\mathsf{FE}})$.*

Moreover, we say CFPKFE *is sub-exponentially* μ*-secure if* $\mathsf{negl}(\lambda) = O(\exp(-\lambda^{\Omega(1)}))$.

Definition 36 (Sublinear Efficiency of CFPKFE)**.** *We say that* CFPKFE *satisfies sublinearity if there exists a constant* $\epsilon \in (0,1)$ *and a fixed polynomial* poly *such that for all polynomial* $n_{\mathsf{FE}}(\lambda), m_{\mathsf{FE}}(\lambda)$*, any* crs_i*'s in the support of* PPGen*, and any* $\mathsf{PK}, \mathsf{MergedCrs}$ *in the support of* Setup*, the size of the circuit computing* $\mathsf{Enc}(\mathsf{PK}, \cdot, \mathsf{MergedCrs})$ *is* $O(m_{\mathsf{FE}}^{1-\epsilon} \cdot \mathsf{poly}(\lambda, n_{\mathsf{FE}}))$*, and moreover this circuit is uniformly efficiently generatable. If we are working with the function class* $\mathcal{F}_{\mathsf{FE}}$ *instead of* $\mathcal{F}_{\mathsf{CFPKFE}}$*, we require the size of the circuit computing* $\mathsf{Enc}(\mathsf{PK}, \cdot, \mathsf{MergedCrs})$ *to be* $O(s_{\mathsf{FE}}^{1-\epsilon} \cdot \mathsf{poly}(\lambda, n_{\mathsf{FE}}))$*.*

We first follow the construction of [JLS22] to show that there exists sublinear CFPKFE for $\mathcal{F}_{\mathsf{CFPKFE}}$.

Theorem 7.2. *Assume that for all* $B = O(1)$*, there exist constants* $c_1, c_2, c_3 > 0$ *and* $d \in \mathbb{N}$ *such that for all* $\tau > 0$*, there exists a* (c_1, c_2, c_3, τ)*-sublinear* CFPRE *with degree* d *encoding and (sub-exponential)* μ*-security. Assume additionally the existence of a (sub-exponentially) secure* PHFE *for degree* d*.*

Then for all $B = O(1)$*, there exists a (sub-exponentially)* μ*-secure* CFPKFE *for* $\mathcal{F}_{\mathsf{CFPKFE}, n_{\mathsf{FE}}, m_{\mathsf{FE}}, \lambda}$ *with sublinear efficiency.*

Applying Theorems 7.1 and 6.2 yields the following corollary:

Corollary 7.3. *Assuming (sub-exponential)* LPN *over large fields, (sub-exponential)* $\mathsf{SparseLPN}$ *over* $\mathbb{Z}_2$*, and (sub-exponential)* DLIN *on symmetric bilinear groups of prime order, for all* $B = O(1)$*, there exists a (sub-exponential)* $\Omega(1)$*-secure* CFPKFE *for* $\mathcal{F}_{\mathsf{CFPKFE}, n_{\mathsf{FE}}, m_{\mathsf{FE}}, \lambda}$*.*

As noted in Remark 7.3 of [JLS22], using Yao's garbled circuits [Yao86] allows us to bootstrap this to sublinear CFPKFE for $\mathcal{F}_{\mathsf{FE}}$ [AJS15]. We hence restrict attention to the function class $\mathcal{F}_{\mathsf{FE}}$ going forward.

Theorem 7.4 ([AJS15])**.** *Suppose that for all* $B = O(1)$*, there exists sublinear (sub-exponentially)* μ*-secure* CFPKFE *for* $\mathcal{F}_{\mathsf{CFPKFE}, n_{\mathsf{FE}}, m_{\mathsf{FE}}, \lambda}$*. Then, for all* $B = O(1)$*, there exists sublinear (sub-exponentially)* μ*-secure* CFPKFE *for* $\mathcal{F}_{\mathsf{FE}, n_{\mathsf{FE}}, s_{\mathsf{FE}}, \lambda}$*.*

7.2.1 Construction Details

We will essentially use the CFPRE under the hood of PHFE [Wee20]. We first specify parameters and ingredients. Let $c_1, c_2, c_3 > 0$ and $d \in \mathbb{N}$ be efficiency parameters inherited from the CFPRE as specified in Theorem 6.2. We assume without loss of generality that $c_2 > 1$ for simplicity.

1. For the CFPRE scheme, we set:
   - $n_{\mathsf{CFPRE}} = n_{\mathsf{FE}}$,
   - $m_{\mathsf{CFPRE}} = m_{\mathsf{FE}}^{1-\delta}$ and $k_{\mathsf{CFPRE}} = m_{\mathsf{FE}}^{\delta}$, where $\delta \in (0,1)$ is a parameter such that $(1-c_1)(1-\delta) + \delta c_2 < 1$ and $1 - \delta + \delta(1 - c_3) < 1$ (where the last inequality trivially holds because $\delta, c_3 \in (0,1)$). Explicitly, we can set

$$\delta = \frac{1}{2} \cdot \frac{c_1}{c_1 + c_2 - 1}.$$

(Note that since we have assumed $c_2 > 1$, we in fact have $\delta < 1/2$.) Indeed, since we have $\lambda^\delta \leq m_{\mathsf{FE}}^\delta = k_{\mathsf{CFPRE}}$, we have k_{CFPRE} lower-bounded by a fixed polynomial in λ, as needed for CFPRE. Moreover, since $n_{\mathsf{FE}} \leq m_{\mathsf{FE}} = k_{\mathsf{CFPRE}}^{1/\delta}$, both $m_{\mathsf{CFPRE}} = m_{\mathsf{FE}}^{1-\delta}$ and $n_{\mathsf{CFPRE}} = n_{\mathsf{FE}}$ are upper-bounded by fixed polynomials in k_{CFPRE}, as needed for CFPRE.

- $\tau > 0$ sufficiently small such that $(1-\delta)(1+\tau) + \delta(1-c_3) < 1$. Explicitly, we can set

$$\tau = \frac{1}{2} \cdot \frac{\delta c_3}{1-\delta}.$$

  We crucially here rely on the order of quantifiers in Theorem 6.2; there exists a fixed choice of c_1, c_2, c_3, d such that we can set τ to be arbitrarily close to 0.
- Define ℓ_{CFPRE} to be the maximum of the lengths of PI and SI, where $(\mathsf{PI}, \mathsf{SI})$ is the output of PreProc.

2. For the PHFE scheme, we require support for degree $(d, 2)$ polynomials and set $n_{\mathsf{PHFE}} = \ell_{\mathsf{CFPRE}}$.

```
CFPKFE Construction

$\mathsf{CFPKFE.PPGen}(1^\lambda, 1^{n_{\mathsf{FE}}}, m_{\mathsf{FE}})$:
  1. Output $\mathsf{CFPRE.crs} \leftarrow \mathsf{CFPRE.PPGen}(1^{n_{\mathsf{CFPRE}}}, 1^{m_{\mathsf{CFPRE}}}, 1^{k_{\mathsf{CFPRE}}}, 1^\lambda)$.

$\mathsf{CFPKFE.Setup}(1^\lambda, 1^{n_{\mathsf{FE}}}, m_{\mathsf{FE}}, \{\mathsf{CFPRE.crs}_j\}_{j\in[B]})$:
  1. Sample $(\mathsf{PHFE.crs}, p) \leftarrow \mathsf{PHFE.PPGen}(1^\lambda)$.
  2. Sample $(\mathsf{PHFE.PK}, \mathsf{PHFE.MSK}) \leftarrow \mathsf{PHFE.Setup}(d, 1^{n_{\mathsf{PHFE}}}, \mathsf{PP})$.
  3. Run $\mathsf{CFPRE.MergedCrs} \leftarrow \mathsf{CFPRE.Setup}(1^{n_{\mathsf{CFPRE}}}, 1^{m_{\mathsf{CFPRE}}}, 1^{k_{\mathsf{CFPRE}}}, 1^\lambda, \{\mathsf{CFPRE.crs}_j\}_{j\in[B]})$.
  4. Output $\mathsf{PK} = (\mathsf{PHFE.PK}, \mathsf{PHFE.crs}, p)$, $\mathsf{MSK} = \mathsf{PHFE.MSK}$, and $\mathsf{MergedCrs} = \mathsf{CFPRE.MergedCrs}$.

$\mathsf{CFPKFE.Enc}(\mathsf{PK}, \mathbf{x} \in \{0,1\}^{n_{\mathsf{FE}}}, \mathsf{MergedCrs} = \mathsf{CFPRE.MergedCrs})$:
  1. Parse $\mathsf{PK} = (\mathsf{PHFE.PK}, \mathsf{PHFE.crs}, p)$.
  2. Preprocess $\mathbf{x}$ using the CFPRE scheme:
     $(\mathsf{PI}, \mathsf{SI}) \leftarrow \mathsf{CFPRE.PreProc}(1^{k_{\mathsf{CFPRE}}}, 1^\lambda, n_{\mathsf{CFPRE}}, m_{\mathsf{CFPRE}}, p, \mathbf{x}, \mathsf{CFPRE.MergedCrs})$.
  3. Output $\mathsf{CT} \leftarrow \mathsf{PHFE.Enc}(\mathsf{PHFE.PK}, (\mathsf{PI}, \mathsf{SI}))$.

$\mathsf{CFPKFE.KeyGen}(\mathsf{MSK}, f, \mathsf{MergedCrs} = \mathsf{CFPRE.MergedCrs})$:
  1. Let $f_1, \ldots, f_T$ be degree $(d, 2)$ polynomials computing $\mathsf{CFPRE.Encode}(f, (\cdot,\cdot), \mathsf{CFPRE.MergedCrs})$.
  2. For each $i \in [T]$, compute $\mathsf{SK}_i \leftarrow \mathsf{PHFE.KeyGen}(\mathsf{PHFE.MSK}, f_i)$.
  3. Output $\mathsf{SK}_f = (\mathsf{SK}_1, \ldots, \mathsf{SK}_T)$.

$\mathsf{CFPKFE.Dec}(\mathsf{SK}_f, \mathsf{CT}, \mathsf{MergedCrs} = \mathsf{CFPRE.MergedCrs})$:
  1. Parse $\mathsf{SK}_f = (\mathsf{SK}_1, \ldots, \mathsf{SK}_T)$. For each $i \in [T]$, compute $y_i \leftarrow \mathsf{PHFE.Dec}(\mathsf{SK}_i, \mathsf{CT})$.
  2. Let $\mathbf{y} = (y_1, \ldots, y_T)$ and output $\mathsf{CFPRE.Decode}(\mathbf{y}, \mathsf{CFPRE.MergedCrs})$.
```

Figure 7: Our CFPKFE Construction.

Our construction is described in Figure 7. We now argue that it satisfies all the necessary properties:

Correctness: Correctness is straightforward from the correctness of the underlying CFPRE and PHFE schemes. By correctness of the PHFE, we will have for each $i \in [T]$ that $y_i = f_i(\mathsf{PI}, \mathsf{SI})$. In other words, we will have $\mathbf{y} = \mathsf{CFPRE.Encode}(f, (\mathsf{PI}, \mathsf{SI}), \mathsf{MergedCrs})$. It now follows from the correctness of the CFPRE that we will have $\mathsf{CFPRE.Decode}(\mathbf{y}, \mathsf{MergedCrs}) = f(\mathbf{x})$, implying correctness of our scheme.

Security: We will take $\mathsf{GoodPPGen}$ to sample from $\mathsf{CFPRE.GoodPPGen}(1^{n_{\mathsf{CFPRE}}}, 1^{m_{\mathsf{CFPRE}}}, 1^{k_{\mathsf{CFPRE}}}, 1^{\lambda})$ and similarly for $\mathsf{BadPPGen}$. With this in mind, we prove security for this choice of $\mathsf{GoodPPGen}$ and $\mathsf{BadPPGen}$. We will provide a series of hybrid games for a fixed string $\mathsf{good} \in \{0,1\}^B$ with at least one entry being 1, starting from the CFPKFE security game in Definition 35 and ending with a game that does not depend at all on the challenger's bit b:

Hybrid_0: This will be exactly the security game in Definition 35.

Hybrid_1: Instead of $\mathsf{PHFE.Setup}$, $\mathsf{PHFE.Enc}$, and $\mathsf{PHFE.KeyGen}$, the challenger will use $\mathsf{PHFE.}\widetilde{\mathsf{Setup}}$, $\mathsf{PHFE.}\widetilde{\mathsf{Enc}}$, $\mathsf{PHFE.}\widetilde{\mathsf{KeyGen}}$ from the simulation security property of PHFE specified in Definition 31. The challenger can evaluate $\mathsf{PHFE.}\widetilde{\mathsf{Enc}}$ since the adversary provides $\mathbf{x}_b$ and then the challenger will compute $(\mathsf{PI}, \mathsf{SI})$. The challenger can also evaluate $\mathsf{PHFE.}\widetilde{\mathsf{KeyGen}}$ given a function query f since they can compute $\mathsf{CFPRE.Encode}(f, (\mathsf{PI}, \mathsf{SI}), \mathsf{MergedCrs}) = (f_1(\mathsf{PI}, \mathsf{SI}), \ldots, f_T(\mathsf{PI}, \mathsf{SI}))$.

Computational indistinguishability holds due to the simulation security of PHFE, even with $\mathsf{poly}(\lambda)$ function queries that can adaptively depend on the ciphertext. (We note as in Remark 5.2 that this holds even though $\mathsf{GoodPPGen}$ and $\mathsf{BadPPGen}$ may not be efficient.) Note that we incur an additive loss of $\mathsf{negl}_1(\lambda)$ (as in Definition 31) due to the randomness over $\mathsf{PHFE.PPGen}$. Note importantly that since $\mathsf{PHFE.}\widetilde{\mathsf{Enc}}$ does not use SI, SI no longer appears anywhere in this hybrid except for the challenger's computation of $\mathsf{CFPRE.Encode}(f, (\mathsf{PI}, \mathsf{SI}), \mathsf{MergedCrs})$ when evaluating $\mathsf{PHFE.}\widetilde{\mathsf{KeyGen}}$.

Hybrid_2: Now, the challenger will always compute

$$(\mathsf{PI}, \mathsf{SI}) \leftarrow \mathsf{CFPRE.PreProc}(1^{k_{\mathsf{CFPRE}}}, 1^{\lambda}, n_{\mathsf{CFPRE}}, m_{\mathsf{CFPRE}}, p, \mathbf{x}_0, \mathsf{CFPRE.MergedCrs}),$$

rather than

$$(\mathsf{PI}, \mathsf{SI}) \leftarrow \mathsf{CFPRE.PreProc}(1^{k_{\mathsf{CFPRE}}}, 1^{\lambda}, n_{\mathsf{CFPRE}}, m_{\mathsf{CFPRE}}, p, \mathbf{x}_b, \mathsf{CFPRE.MergedCrs}),$$

before sampling $\widetilde{\mathsf{CT}} \leftarrow \mathsf{PHFE.}\widetilde{\mathsf{Enc}}(\mathsf{PHFE.}\widetilde{\mathsf{MSK}}, \mathsf{PI})$ and sending $\widetilde{\mathsf{CT}}, \widetilde{\mathsf{PK}} = (\mathsf{PHFE.}\widetilde{\mathsf{PK}}, \mathsf{crs}, p), \mathsf{MergedCrs}$ to the adversary.

This computational indistinguishability holds due to security of the CFPRE scheme, noting that at least one entry of $\mathsf{good} \in \{0,1\}^B$ has a 1: the adversary sends $\mathbf{x}_0, \mathbf{x}_1$, the challenger replies with $\mathsf{MergedCrs}$ and a function of PI where

$$(\mathsf{PI}, \mathsf{SI}) \leftarrow \mathsf{CFPRE.PreProc}(1^{k_{\mathsf{CFPRE}}}, 1^{\lambda}, n_{\mathsf{CFPRE}}, m_{\mathsf{CFPRE}}, p, \mathbf{x}_b, \mathsf{CFPRE.MergedCrs}),$$

the adversary replies with a function f, and the challenger replies with a function of

$$\mathsf{CFPRE.Encode}(f, (\mathsf{PI}, \mathsf{SI}), \mathsf{MergedCrs}),$$

with no other dependence on SI. Since $f(\mathbf{x}_0) = f(\mathbf{x}_1)$, the claimed indistinguishability follows.

Since Hybrid_2 does not depend on the bit b anymore, the security of our CFPKFE follows.

Sublinear Efficiency: Let T_{CFPRE} denote the size of the circuit computing $\mathsf{CFPRE.PreProc}(1^{k_{\mathsf{CFPRE}}}, 1^\lambda, n_{\mathsf{CFPRE}}, m_{\mathsf{CFPRE}}, p, \cdot, \mathsf{CFPRE.MergedCrs})$. We then trivially have $n_{\mathsf{PHFE}} = \ell_{\mathsf{CFPRE}} \leq T_{\mathsf{CFPRE}}$. By the linear efficiency of the PHFE, the size of the circuit computing $\mathsf{PHFE.Enc}(\mathsf{PHFE.PK}, \cdot)$ is $O(n_{\mathsf{PHFE}} \cdot \mathsf{poly}(\lambda)) \leq O(T_{\mathsf{CFPRE}} \cdot \mathsf{poly}(\lambda))$. It hence follows that the overall circuit size of $\mathsf{CFPKFE.Enc}(\mathsf{PK}, \cdot, \mathsf{MergedCrs})$ is bounded above by $O(T_{\mathsf{CFPRE}} \cdot \mathsf{poly}(\lambda))$. The sublinear efficiency of the CFPRE tells us that:

$$\begin{aligned}
T_{\mathsf{CFPRE}} &\leq \left((n_{\mathsf{CFPRE}} + m_{\mathsf{CFPRE}}^{1-c_1}) \cdot k_{\mathsf{CFPRE}}^{c_2} + (n_{\mathsf{CFPRE}} + m_{\mathsf{CFPRE}})^{1+\tau} \cdot k_{\mathsf{CFPRE}}^{1-c_3} \right) \cdot \mathsf{poly}(\lambda, \log p) \\
&\leq \left((n_{\mathsf{FE}} + m_{\mathsf{FE}}^{(1-\delta)(1-c_1)}) \cdot m_{\mathsf{FE}}^{\delta c_2} + (n_{\mathsf{FE}} + m_{\mathsf{FE}}^{1-\delta})^{1+\tau} \cdot m_{\mathsf{FE}}^{\delta(1-c_3)} \right) \cdot \mathsf{poly}(\lambda) \\
&\leq \left((n_{\mathsf{FE}} + m_{\mathsf{FE}}^{(1-\delta)(1-c_1)}) \cdot m_{\mathsf{FE}}^{\delta c_2} + (n_{\mathsf{FE}}^{1+\tau} + m_{\mathsf{FE}}^{(1-\delta)(1+\tau)}) \cdot m_{\mathsf{FE}}^{\delta(1-c_3)} \right) \cdot \mathsf{poly}(\lambda) \\
&\leq \left(n_{\mathsf{FE}}^{1+\tau} \cdot m_{\mathsf{FE}}^{\max(\delta c_2, \delta(1-c_3))} + m_{\mathsf{FE}}^{\max((1-\delta)(1-c_1)+\delta c_2, (1-\delta)(1+\tau)+\delta(1-c_3))} \right) \cdot \mathsf{poly}(\lambda) \\
&\leq \left(m_{\mathsf{FE}}^{\max((1-\delta)(1-c_1)+\delta c_2, (1-\delta)(1+\tau)+\delta(1-c_3))} \right) \cdot \mathsf{poly}(n_{\mathsf{FE}}, \lambda).
\end{aligned}$$

(Since we trivially have $\delta c_2 < (1-\delta)(1-c_1) + \delta c_2$ and $\delta(1-c_3) < (1-\delta)(1+\tau) + \delta(1-c_3)$, we drop these terms in the exponent.) By our choice of δ, we have $(1-\delta)(1-c_1) + \delta c_2 < 1$. By our choice of τ, we have $(1-\delta)(1+\tau) + \delta(1-c_3) < 1$. Therefore, the exponent on m_{FE} is $1 - \Omega(1)$, so the circuit size of $\mathsf{CFPKFE.Enc}(\mathsf{PK}, \cdot, \mathsf{MergedCrs})$ is at most

$$T_{\mathsf{CFPRE}} \cdot \mathsf{poly}(\lambda) \leq m_{\mathsf{FE}}^{1-\Omega(1)} \cdot \mathsf{poly}(n_{\mathsf{FE}}, \lambda).$$

This completes the proof of Theorem 7.2.

7.3 Secret-Key Functional Encryption

Here, we define secret-key functional encryption and trivially migrate the CFPKFE scheme we just constructed to the secret-key setting. The reason for doing this is that our construction is still insecure with probability $1 - \Omega(1)$. In Section 7.4, we will amplify this to a construction that is sub-exponentially secure (which is necessary for $i\mathcal{O}$), for which we will rely on the *secret-key* FE combiner due to [JMS20]. From here on, we will only need to work with the function class $\mathcal{F}_{\mathsf{FE}}$.

Definition 37 (CFSKFE Syntax and Correctness)**.** *A secret-key functional encryption scheme* CFSKFE *for the function class* $\mathcal{F}_{\mathsf{FE}, n_{\mathsf{FE}}, m_{\mathsf{FE}}, \lambda}$ *has the same syntax as* CFPKFE*, with the following changes:*

- PPGen *will take* $1^\lambda, 1^{n_{\mathsf{FE}}}, s_{\mathsf{FE}}$ *as input and output a* crs.
- Setup *will take* $1^\lambda, 1^{n_{\mathsf{FE}}}, s_{\mathsf{FE}}, \{\mathsf{crs}_j\}_{j \in [B]}$ *as input and output a secret key* MSK *and* $\mathsf{MergedCrs}$.
- Enc *must use* MSK *instead of* PK.

The correctness property is exactly analogous to that of CFPKFE.

Definition 38 (CFSKFE with Inefficient PPGen)**.** *We say that* CFSKFE *is a* CFSKFE *with inefficient* PPGen *if we no longer require* PPGen *to have a polynomial-size circuit. In other words,* PPGen *just needs to sample from some distribution over* $\{0,1\}^*$.

While an inefficient-PPGen CFSKFE is not directly useful towards constructing $i\mathcal{O}$, we will use this notion for technical reasons in the security proof when bootstrapping to truly sub-exponentially secure CFSKFE in Section 7.4.

Definition 39 (CFSKFE Security)**.** *We say* CFSKFE *is (selective) IND μ-secure (for some function $\mu(\lambda)$) if there exist (possibly inefficient) algorithms* $\mathsf{GoodPPGen}$ *and* $\mathsf{BadPPGen}$ *with the same syntax as* PPGen *such that both of the following hold:*

1. *For all stateful p.p.t. adversaries $\mathcal{A}$ and binary strings* $\mathsf{good} \in \{0,1\}^B$ *such that at least one entry of* good *is 1, there exists a negligible function* negl *such that $\mathcal{A}$ succeeds in the below game with probability at most $1/2 + \mathsf{negl}(\lambda)$:*
   - *$\mathcal{A}$ takes as input 1^λ and chooses an input length n_{FE} and upper bound on the circuit size s_{FE}. They then provide queries $\left\{(\mathbf{x}_0^i, \mathbf{x}_1^i)\right\}_{i\in[Q_{\mathsf{Enc}}]}$ to the challenger, where $Q_{\mathsf{Enc}}(\lambda)$ is some a priori unbounded polynomial. We require that $\mathbf{x}_0^i, \mathbf{x}_1^i \in \{0,1\}^{n_{\mathsf{FE}}}$ for all i.*
   - *For each $i \in [B]$, the challenger samples $\mathsf{crs}_i \leftarrow \mathsf{GoodPPGen}(1^\lambda, 1^{n_{\mathsf{FE}}}, s_{\mathsf{FE}})$ if $\mathsf{good}_i = 1$, and $\mathsf{BadPPGen}(1^\lambda, 1^{n_{\mathsf{FE}}}, s_{\mathsf{FE}})$ otherwise.*
   - *The challenger samples $b \leftarrow \{0,1\}$ and $\mathsf{MSK}, \mathsf{MergedCrs} \leftarrow \mathsf{Setup}(1^\lambda, 1^{n_{\mathsf{FE}}}, s_{\mathsf{FE}}, \{\mathsf{crs}_j\}_{j\in[B]})$. They then compute $\mathsf{CT}_i \leftarrow \mathsf{Enc}(\mathsf{MSK}, \mathbf{x}_b^i, \mathsf{MergedCrs})$ for all $i \in [Q_{\mathsf{Enc}}]$. They send all the CT_i's back to $\mathcal{A}$, along with $\mathsf{MergedCrs}$.*
   - *$\mathcal{A}$ submits a function query $f \in \mathcal{F}_{\mathsf{FE},n_{\mathsf{FE}},s_{\mathsf{FE}},\lambda}$ to the challenger. The challenger checks that $f(\mathbf{x}_0^i) = f(\mathbf{x}_1^i)$ for all $i \in [Q_{\mathsf{Enc}}]$. If not, they reject immediately. Otherwise, they send $\mathcal{A}$ the function key $\mathsf{SK}_f \leftarrow \mathsf{KeyGen}(\mathsf{MSK}, f, \mathsf{MergedCrs})$.*
   - *The adversary outputs a guess $b' \in \{0,1\}$ and wins if and only if $b = b'$ (and the challenger did not reject already).*
2. *We have $\mu(\lambda) = \Omega(1)$ and the following two distributions are identical:*
   - *Sample* $\mathsf{crs} \leftarrow \mathsf{PPGen}(1^\lambda, 1^{n_{\mathsf{FE}}}, s_{\mathsf{FE}})$.
   - *With probability μ, sample* $\mathsf{crs} \leftarrow \mathsf{GoodPPGen}(1^\lambda, 1^{n_{\mathsf{FE}}}, s_{\mathsf{FE}})$. *With probability $1-\mu$, sample* $\mathsf{crs} \leftarrow \mathsf{BadPPGen}(1^\lambda, 1^{n_{\mathsf{FE}}}, s_{\mathsf{FE}})$.

Moreover, we say CFSKFE *is sub-exponentially μ-secure if* $\mathsf{negl}(\lambda) = O(\exp(-\lambda^{\Omega(1)}))$.
Moreover, for $\mu = 1$, we simply say that CFSKFE *is secure (or sub-exponentially secure).*

We use the notation SKFE to denote a CFSKFE with $\mu = 1$ and $B = 1$, which corresponds to the standard notion of (single-key, sublinear efficient) secret-key functional encryption, without consideration for combiners.

Definition 40 (Sublinear Efficiency of CFSKFE)**.** *We say that* CFSKFE *satisfies sublinearity if there exists a constant $\epsilon \in (0,1)$ and a fixed polynomial* poly *such that for all polynomial $n_{\mathsf{FE}}(\lambda), s_{\mathsf{FE}}(\lambda)$, any* crs_i*'s in the support of* PPGen*, and any* $\mathsf{MSK}, \mathsf{MergedCrs}$ *in the support of* Setup*, the size of the circuit computing* $\mathsf{Enc}(\mathsf{MSK}, \cdot, \mathsf{MergedCrs})$ *is $O(s_{\mathsf{FE}}^{1-\epsilon} \cdot \mathsf{poly}(\lambda, n_{\mathsf{FE}}))$, and this circuit is uniformly efficiently generatable.*

Unsurprisingly, this is a weaker notion than the CFPKFE we just constructed. We verify this in the following lemma:

Lemma 7.5. *Assume for all $B = O(1)$, there exists a μ-secure* CFPKFE *for* $\mathcal{F}_{\mathsf{FE},n_{\mathsf{FE}},s_{\mathsf{FE}},\lambda}$. *Then for all $B = O(1)$, there exists a μ-secure* CFSKFE *for* $\mathcal{F}_{\mathsf{FE},n_{\mathsf{FE}},s_{\mathsf{FE}},\lambda}$. *Moreover, if the* CFPKFE *is sub-exponentially secure, so is the* CFSKFE.

Proof. The construction is straightforward; we simply include the public key of the CFPKFE in the MSK of the CFSKFE:

- PPGen will simply sample $\mathsf{crs} \leftarrow \mathsf{CFPKFE.PPGen}$.
- Setup will sample $(\mathsf{CFPKFE.PK}, \mathsf{CFPKFE.MSK}, \mathsf{CFPKFE.MergedCrs}) \leftarrow \mathsf{CFPKFE.Setup}$ and output $\mathsf{MSK} = (\mathsf{CFPKFE.PK}, \mathsf{CFPKFE.MSK})$ and $\mathsf{MergedCrs} = \mathsf{CFPKFE.MergedCrs}$.
- $\mathsf{Enc}(\mathsf{MSK} = (\mathsf{CFPKFE.PK}, \mathsf{CFPKFE.MSK}), \mathbf{x}, \mathsf{MergedCrs})$ will output $\mathsf{CT} \leftarrow \mathsf{CFPKFE.Enc}(\mathsf{CFPKFE.PK}, \mathbf{x}, \mathsf{MergedCrs})$.
- $\mathsf{KeyGen}(\mathsf{MSK} = (\mathsf{CFPKFE.PK}, \mathsf{CFPKFE.MSK}), f, \mathsf{MergedCrs})$ will output $\mathsf{SK}_f \leftarrow \mathsf{CFPKFE.KeyGen}(\mathsf{CFPKFE.MSK}, f, \mathsf{MergedCrs})$.
- $\mathsf{Dec}(\mathsf{SK}_f, \mathsf{CT}, \mathsf{MergedCrs})$ will output $\mathsf{CFPKFE.Dec}(\mathsf{SK}_f, \mathsf{CT}, \mathsf{MergedCrs})$.

Correctness is clear. Sublinearity also directly follows from the sublinearity of the CFPKFE. It remains to address security. We will take $\mathsf{GoodPPGen}$ to be $\mathsf{CFPKFE.GoodPPGen}$ and $\mathsf{BadPPGen}$ to be $\mathsf{CFPKFE.BadPPGen}$. We want to show that the cases where $b = 0$ and $b = 1$ in the CFSKFE security game are indistinguishable to a p.p.t. adversary $\mathcal{A}$. For each $j \in [0, Q_{\mathsf{Enc}}]$, define Hybrid_j to be the same security game, except the challenger samples ciphertexts as follows: if $i \leq j$, they sample $\mathsf{CT}_i \leftarrow \mathsf{Enc}(\mathsf{MSK}, \mathbf{x}_0^i, \mathsf{MergedCrs})$. Otherwise, they sample $\mathsf{CT}_i \leftarrow \mathsf{Enc}(\mathsf{MSK}, \mathbf{x}_1^i, \mathsf{MergedCrs})$. Since Hybrid_0 corresponds to the $b = 1$ case and $\mathsf{Hybrid}_{Q_{\mathsf{Enc}}}$ corresponds to the $b = 0$ case, it suffices to check for each $j \in [Q_{\mathsf{Enc}}]$ that Hybrid_{j-1} and Hybrid_j are indistinguishable to $\mathcal{A}$. There are only polynomially many hybrids, so the negligible advantage will be preserved.

Suppose for the sake of contradiction that there exists j and $\mathcal{A}$ so that $\mathcal{A}$ distinguishes Hybrid_{j-1} and Hybrid_j. The only difference between these two hybrids is whether the challenger constructs CT_j by encrypting $\mathbf{x}_0^j$ or $\mathbf{x}_1^j$. So one can now break the security of the CFPKFE game as follows: act as the challenger for $\mathcal{A}$. For all $i < j$, sample $\mathsf{CT}_i \leftarrow \mathsf{CFPKFE.Enc}(\mathsf{CFPKFE.PK}, \mathbf{x}_0^i, \mathsf{MergedCrs})$. For all $i > j$, sample $\mathsf{CT}_i \leftarrow \mathsf{CFPKFE.Enc}(\mathsf{CFPKFE.PK}, \mathbf{x}_1^i, \mathsf{MergedCrs})$. Finally, send $\mathbf{x}_0^j$ and $\mathbf{x}_1^j$ to the CFPKFE challenger and feed the ciphertext they return as CT_j to $\mathcal{A}$. If, after a function query, $\mathcal{A}$ can guess which hybrid it is in, our algorithm will have broken the security of the CFPKFE. This completes our proof.

□

7.4 CFSKFE Combiners

The CFSKFE we have just constructed has one central problem that we will need to rectify before being able to bootstrap it to $i\mathcal{O}$ using results by [KNT22]: it is only sub-exponentially secure with probability $\mu = \Omega(1)$ over the randomness of PPGen. To address these, we use the notion of a secret-key functional encryption combiner, as defined and constructed by [ABJ+19, JMS20]. Ideally, this allows us to instantiate polynomially many instances of our current CFSKFE scheme, and combine them into a truly secure SKFE scheme as long as at least one instance is secure. This will be true with all but sub-exponential probability, so this will give us the desired CFSKFE.

However, their combiner does not preserve sublinear efficiency, as the 3-nesting in their construction does not preserve sublinear efficiency. Therefore, instead of assuming that just one of the CFSKFE schemes is secure, we will use the combiner-friendliness of our primitives to insert some correlated 3-nesting combinatorial structure among the CFSKFE candidates. The theorem shown by [JMS20] is the following:

Theorem 7.6 (Informal, [JMS20])**.** *Suppose we have r^3 candidate CFSKFE schemes $\{\mathsf{CFSKFE}_{i_1,i_2,i_3}\}_{i_1,i_2,i_3\in[r]}$ for some polynomial $r = r(\lambda)$ bounded independently of n_{FE} and s_{FE} and the function class $\mathcal{F}_{\mathsf{FE}}$ such that all schemes satisfy correctness and sublinear efficiency, and there exists $i^* \in [r]$ such that $\{\mathsf{CFSKFE}_{i^*,i_2,i_3}\}_{i_2,i_3\in[r]}$, $\{\mathsf{CFSKFE}_{i_1,i^*,i_3}\}_{i_1,i_3\in[r]}$, $\{\mathsf{CFSKFE}_{i_1,i_2,i^*}\}_{i_1,i_2\in[r]}$ are all secure. Here, we allow PPGen to be correlated across the r^3 candidates.*

Then there exists a uniform compiler that takes $\{\mathsf{CFSKFE}_{i_1,i_2,i_3}\}_{i_1,i_2,i_3\in[r]}$ as input and produces a SKFE that is correct, secure, and satisfies sublinear efficiency. Moreover, security holds even when all candidates have inefficient PPGen. Further, if the security condition stated above for the candidates is sub-exponentially secure, then so is SKFE.

Proof Sketch. Let FEComb denote the SKFE built from the underlying CFSKFE candidates. This theorem follows from [JMS20, Theorem 8], but substituting the 3-nested candidates directly with each $\mathsf{CFSKFE}_{i_1,i_2,i_3}$ and with six extra things to show:

1. We want the combiner to preserve *perfect* correctness, while the combiner in [JMS20] is stated as preserving $1 - \mathsf{negl}(\lambda)$ correctness.

2. In our candidate schemes, we include an additional PPGen algorithm to sample a crs. We need to ensure that the combiner is compatible with this syntax.

3. We use the combiner for CFSKFE candidates that are secure only for single secret key queries, and we will get out FEComb that is also secure for single secret key queries. On the other hand, [JMS20] state their combiner to handle an unbounded polynomial number of function secret key queries, in both the candidates and the combined scheme.

4. We need to show that security holds even when the candidates have inefficient PPGen.

5. Formally, the combiner resulted is stated for polynomially-secure FE candidates (i.e., if one of the inputs is polynomially secure, then the combined one is polynomially secure). We need to ensure the combiner preserves sub-exponential security too.

6. The combiner preserves sublinearity, in the sense that if all CFSKFE candidates satisfy sublinear efficiency, then the combined one does too.

Item 1 follows from inspecting the protocol in [JMS20] (in particular, the ability to generate perfect OT correlations from a PRF key) and the underlying MPC protocol it invokes, as given in [GS22].

Item 2 holds in a direct way. In $\mathsf{FEComb.Setup}$, for all $(i_1, i_2, i_3) \in [r]^3$, we will sequentially run $\mathsf{CFSKFE}_{i_1,i_2,i_3}.\mathsf{PPGen}$ B times independently (but possibly correlated across candidates) and then $\mathsf{CFSKFE}_{i_1,i_2,i_3}.\mathsf{Setup}$ for each candidate to produce $\mathsf{MSK}_{i_1,i_2,i_3}$ and $\mathsf{MergedCrs}_{i_1,i_2,i_3}$ for each of the candidates. The output of $\mathsf{FEComb.Setup}$ is then

$$\left(\mathsf{crs} = \{\mathsf{MergedCrs}_{i_1,i_2,i_3}\}_{i_1,i_2,i_3\in[r]}, \mathsf{MSK} = (\{\mathsf{MSK}_{i_1,i_2,i_3}\}_{i_1,i_2,i_3\in[r]}, \mathsf{E.SK})\right),$$

where $\mathsf{E.SK}$ is the secret key for the CPA-secure secret-key encryption scheme used in the combiner. (That is, we no longer need a $\mathsf{FEComb.PPGen}$.)

Item 3 follows from inspection of the construction in [JMS20, Section 6.2], since the 3-nested schemes are now replaced with separate CFSKFE candidates.

Item 4 follows directly from inspecting the security proof in [JMS20, Theorem 8], where the adversary is crucially allowed to use non-uniformity to hard-code values of crs in the hybrid argument.

Item 5 follows directly assuming the existence of sub-exponentially secure one-way functions (which is guaranteed by universal one-way function constructions) and sub-exponentially secure combiner-friendly homomorphic secret sharing (CFHSS) schemes, which exist from sub-exponentially secure one-way functions [JMS20, Theorem 7].

Item 6 requires a calculation. Unfortunately, the 3-nesting construction of FE given in [ABJ+19, Theorem 10], [JMS20, Theorem 9] does not preserve sublinearity, since the decryption circuit has no sublinearity condition. As such, we circumvent this by assuming the r^3 candidates already have a 3-nested combinatorial structure baked in, as stated in the hypothesis of the theorem.

For the circuits C handled in FEComb, let n_{FE} be the input length, and let s_{FE} be the size of the circuit. Let $n'_{\mathsf{FE}}, s'_{\mathsf{FE}}$ be the analogous parameters for the circuits handled by each of the r^3 CFSKFE candidates. Each of the candidates uses KeyGen for a circuit named H_{i_1,i_2,i_3}, using the notation directly from [JMS20, Section 6.2].

The input length to said circuit, namely n'_{FE}, can be readily seen to equal to the length of a combiner-friendly homomorphic secret sharing (CFHSS) share s_{i_1,i_2,i_3}, plus the length of a CPA-secure secret-key encryption (SKE) secret key (namely, λ), plus 1. The length of s_{i_1,i_2,i_3} can be upper-bounded by $3n_{\mathsf{FE}} + 6\lambda$, since for each $(i_1, i_2, i_3) \in [r]^3$, the share contains at most 3 XOR secret shares of length n_{FE}, and 6 PRF keys. Therefore,

$$n'_{\mathsf{FE}} \leq 3n_{\mathsf{FE}} + 6\lambda + \lambda + 1 = O(n_{\mathsf{FE}} + \lambda).$$

For s'_{FE}, the size of H_{i_1,i_2,i_3} can be upper bounded by the size of the SKE decryption circuit plus the size of a CFHSS function encoding C_{i_1,i_2,i_3} (plus lower order terms for the wiring performing the "if" condition). As mentioned in [JMS20, Theorem 7], instantiating the underlying conforming MPC protocol with [GMW87], we have $|C_{i_1,i_2,i_3}| \leq s_{\mathsf{FE}} \cdot \mathsf{poly}(\lambda, r)$. For the size of the decryption circuit, we can instantiate the encryption with the standard pseudorandom one-time pad. For messages of length ℓ, the decryption circuit size will be dominated by the circuit size of PRF evaluation for functions mapping λ bits to ℓ bits. Using standard length extension of PRGs, this can be made to be $\ell \cdot \mathsf{poly}(\lambda)$. The length ℓ of the messages is the output length of C_{i_1,i_2,i_3}, which is in particular bounded by $|C_{i_1,i_2,i_3}| \leq s_{\mathsf{FE}} \cdot \mathsf{poly}(\lambda, r)$. Therefore, the size of the decryption circuit is

$$O(\ell \cdot \mathsf{poly}(\lambda)) \leq s_{\mathsf{FE}} \cdot \mathsf{poly}(\lambda, r).$$

Thus, we have the bound

$$\begin{aligned} s'_{\mathsf{FE}} = |H_{i_1,i_2,i_3}| \leq s_{\mathsf{FE}} \cdot \mathsf{poly}(\lambda, r) + O(|C_{i_1,i_2,i_3}|) &\leq s_{\mathsf{FE}} \cdot \mathsf{poly}(\lambda, r) + s_{\mathsf{FE}} \cdot \mathsf{poly}(\lambda, r) \\ &\leq s_{\mathsf{FE}} \cdot \mathsf{poly}(\lambda, r). \end{aligned}$$

Now, we can apply succinctness of the underlying FE candidates to say that

$$\begin{aligned} |\mathsf{CFSKFE}_{i_1,i_2,i_3}.\mathsf{Enc}(\mathsf{MSK}, \cdot, \mathsf{crs})| \leq (s'_{\mathsf{FE}})^{1-\Omega(1)} \cdot \mathsf{poly}(n'_{\mathsf{FE}}, \lambda) &\leq (s_{\mathsf{FE}} \cdot \mathsf{poly}(\lambda, r))^{1-\Omega(1)} \cdot \mathsf{poly}(n_{\mathsf{FE}}, \lambda) \\ &\leq s_{\mathsf{FE}}^{1-\Omega(1)} \cdot \mathsf{poly}(n_{\mathsf{FE}}, \lambda), \end{aligned}$$

using the fact that r is some fixed $\mathsf{poly}(\lambda)$.

Lastly, to finish bounding the size of $\mathsf{FEComb.Enc}(\mathsf{MSK}, \cdot, \mathsf{crs})$, we need to bound the size of $\mathsf{CFHSS.InpEncode}(1^\lambda, 1^r, \cdot)$, as we have the bound

$$\begin{aligned}|\mathsf{FEComb.Enc}(\mathsf{MSK}, \cdot, \mathsf{crs})| &\leq |\mathsf{CFHSS.InpEncode}(1^\lambda, 1^r, \cdot)| + r^3 \cdot |\mathsf{CFSKFE}_{i_1,i_2,i_3}.\mathsf{Enc}(\mathsf{MSK}, \cdot, \mathsf{crs})| \\ &\leq |\mathsf{CFHSS.InpEncode}(1^\lambda, 1^r, \cdot)| + r^3 \cdot s_{\mathsf{FE}}^{1-\Omega(1)} \cdot \mathsf{poly}(n_{\mathsf{FE}}, \lambda) \\ &\leq |\mathsf{CFHSS.InpEncode}(1^\lambda, 1^r, \cdot)| + s_{\mathsf{FE}}^{1-\Omega(1)} \cdot \mathsf{poly}(n_{\mathsf{FE}}, \lambda).\end{aligned}$$

The size of $\mathsf{CFHSS.InpEncode}(1^\lambda, 1^r, \cdot)$ is readily seen to be $\mathsf{poly}(n_{\mathsf{FE}}, r, \lambda)$, in particular, independent of s_{FE}. Therefore,

$$\begin{aligned}|\mathsf{FEComb.Enc}(\mathsf{MSK}, \cdot, \mathsf{crs})| &\leq |\mathsf{CFHSS.InpEncode}(1^\lambda, 1^r, \cdot)| + s_{\mathsf{FE}}^{1-\Omega(1)} \cdot \mathsf{poly}(n_{\mathsf{FE}}, \lambda) \\ &\leq \mathsf{poly}(n_{\mathsf{FE}}, r, \lambda) + s_{\mathsf{FE}}^{1-\Omega(1)} \cdot \mathsf{poly}(n_{\mathsf{FE}}, \lambda) \\ &\leq s_{\mathsf{FE}}^{1-\Omega(1)} \cdot \mathsf{poly}(n_{\mathsf{FE}}, \lambda),\end{aligned}$$

showing sublinearity of FEComb, as desired. □

For our purposes, we need the following modification of this theorem, which follows from exactly the same construction and argument as [JMS20], although they do not formally state this:

Theorem 7.7 (Follows from [JMS20])**.** *Suppose we have a* CFSKFE *scheme for* $B = 3$ *for the function class* $\mathcal{F}_{\mathsf{FE}}$ *with perfect correctness and sublinear efficiency, that is* μ*-secure for* $\mu = \Omega(1)$*. Then there exists a secure* SKFE *scheme for the function class* $\mathcal{F}_{\mathsf{FE}}$ *with perfect correctness and sublinear efficiency. Moreover, if the underlying* CFSKFE *is sub-exponentially secure, then the resulting* SKFE *scheme is sub-exponentially secure.*

Proof Sketch. We set $r = r(\lambda) = \lambda$. For our candidates, we will have r^3 separate instances of CFSKFE with $B = 3$, but with correlated values of $\{\mathsf{crs}_j\}_{j\in[3]}$ between instances described as follows. Explicitly, for each $i \in [r]$ and $j \in [3]$, sample $\mathsf{crs}_i^{(j)} \leftarrow \mathsf{CFSKFE.PPGen}(1^\lambda, 1^{n_{\mathsf{FE}}}, s_{\mathsf{FE}})$. For $(i_1, i_2, i_3) \in [r]^3$, let $\mathsf{CFSKFE}_{i_1,i_2,i_3}$ be a new instantiation of CFSKFE that inherits all the algorithms of CFSKFE and additionally uses $\left\{\mathsf{crs}_{i_j}^{(j)}\right\}_{j\in[3]}$ as its collection of crs's. However, $\mathsf{CFSKFE.Setup}$ will be run completely independently between all r^3 instances. In other words, there are a total of $3r$ independent values of $\left\{\mathsf{crs}_i^{(j)}\right\}_{i\in[r],j\in[3]}$, used in a correlated way across r^3 CFSKFE instances, but besides this crs correlation, all r^3 instances are independent of each other. We emphasize though that within a given instance $(i_1, i_2, i_3) \in [r]^3$, the *marginal distribution* (i.e., ignoring the other $r^3 - 1$ instances) of $\left\{\mathsf{crs}_{i_j}^{(j)}\right\}_{j\in[3]}$ is jointly i.i.d. across the 3 crs values.

We now invoke security of each CFSKFE to switch to a hybrid where each $\mathsf{crs}_i^{(j)}$ is sampled from the mixture

$$\mu \cdot \mathsf{CFSKFE.GoodPPGen}(1^\lambda, 1^{n_{\mathsf{FE}}}, s_{\mathsf{FE}}) + (1 - \mu) \cdot \mathsf{CFSKFE.BadPPGen}(1^\lambda, 1^{n_{\mathsf{FE}}}, s_{\mathsf{FE}})$$

instead of from $\mathsf{CFSKFE.PPGen}(1^\lambda, 1^{n_{\mathsf{FE}}}, s_{\mathsf{FE}})$. These are identical, by security of CFSKFE. Let $\mathsf{good} \in \{0,1\}^{r\times 3}$ be such that $\mathsf{good}_{i,j} = 1$ if and only if $\mathsf{crs}_i^{(j)}$ is sampled from $\mathsf{CFSKFE.GoodPPGen}(1^\lambda, 1^{n_{\mathsf{FE}}}, s_{\mathsf{FE}})$.

Claim 7.8. *With probability at least* $1 - \exp(-\Omega(\lambda))$*, there exists some* $i^* \in [r]$ *such that for all* $j \in [3]$, $\mathsf{good}_{i^*,j} = 1$.

Proof. By the mixture characterization, we know that each entry of $\mathsf{good}_{i,j}$ is i.i.d. $\mathsf{Bern}(\mu)$. Therefore, for any fixed $i \in [r]$,

$$\Pr[\forall j \in [3], \mathsf{good}_{i,j} = 1] = \mu^3 = \Omega(1)^3 = \Omega(1).$$

Since all entries in good are independent, the probability that this condition fails for all $i \in [r]$ is $(1 - \Omega(1))^r = \exp(-\Omega(\lambda))$, since $r = \lambda$. □

Now, we can invoke the security of each $\{\mathsf{CFSKFE}_{i^*,i_2,i_3}\}_{i_2,i_3\in[r]}$, $\{\mathsf{CFSKFE}_{i_1,i^*,i_3}\}_{i_1,i_3\in[r]}$, and $\{\mathsf{CFSKFE}_{i_1,i_2,i^*}\}_{i_1,i_2\in[r]}$, since by our combinatorial setup, each of these includes $\mathsf{crs}^{(j)}_{i^*}$ for some $j \in [3]$, which we know implies security by Claim 7.8. Therefore, $\{\mathsf{CFSKFE}_{i^*,i_2,i_3}\}_{i_2,i_3\in[r]}$, $\{\mathsf{CFSKFE}_{i_1,i^*,i_3}\}_{i_1,i_3\in[r]}$, and $\{\mathsf{CFSKFE}_{i_1,i_2,i^*}\}_{i_1,i_2\in[r]}$ are all secure. Note that each of these will have inefficient PPGen, corresponding to either $\mathsf{GoodPPGen}(1^\lambda, 1^{n_{\mathsf{FE}}}, s_{\mathsf{FE}})$ if $\mathsf{good}_{i,j} = 1$ and $\mathsf{BadPPGen}(1^\lambda, 1^{n_{\mathsf{FE}}}, s_{\mathsf{FE}})$ if $\mathsf{good}_{i,j} = 0$.

Finally, we invoke Theorem 7.6 with these r^3 candidates (in distribution over good, which defines i^*) to get the desired result. The security loss (beyond that of Theorem 7.6) is only an additive factor of $\exp(-\Omega(\lambda))$ due to Claim 7.8. □

Although the SKFE scheme we have constructed here makes use of a crs, we remark that this can be removed without loss of generality by including the crs inside the MSK and the function keys SK_f.

7.5 Bootstrapping to $i\mathcal{O}$

To bootstrap our construction of SKFE to $i\mathcal{O}$, we use the following theorem by [KNT22]:

Theorem 7.9 (Corollary 10.8 of [KNT22])**.** *Assume there exists single key query sub-exponentially secure* SKFE *for* $\mathcal{F}_{\mathsf{FE},n_{\mathsf{FE}},s_{\mathsf{FE}},\lambda}$ *satisfying sublinearity. Then there exists (sub-exponentially secure)* $i\mathcal{O}$ *for all circuits.*

Proof Sketch. We briefly outline the chain of results that imply this theorem. We note that our sublinearity requirement for $\mathcal{F}_{\mathsf{FE},n_{\mathsf{FE}},s_{\mathsf{FE}},\lambda}$ corresponds exactly to what [KNT22] refers to as "weak succinctness."

As shown by Brakerski and Segev [BS18] and stated by [KNT22, Theorem 3.14], (sub-exponentially secure, single-key, sublinear) message-private SKFE (as in our definition) implies (sub-exponentially secure, single-key, sublinear) function-private SKFE, where [KNT22] notes that the transformation preserves sublinearity.

[KNT22, Theorem 10.7] shows that (sub-exponentially secure, function-private) sublinear, single-key SKFE implies (sub-exponentially secure, function-private) collusion-resistant SKFE.

Lastly, [KNT22, Theorem 7.3] shows that sub-exponentially secure, collusion-resistant SKFE implies (sub-exponentially secure) $i\mathcal{O}$, as desired. □

Corollary 7.10. *Suppose that the following are true:*

- *There is a* $\Omega(1)$*-secure* PRG *in* NC^0 *with linear stretch;*

- *There exist constants $\nu > 0$ and $d \in \mathbb{N}$ such that for any constant $\tau > 0$, there exists a perfectly correct $\Omega(1)$-secure* SPRG *satisfying ν-sublinear efficiency with degree d and τ-local decompression;*

- *The* LPN *assumption over large fields; and*

- *The* DLIN *assumption over prime order symmetric bilinear groups.*

Then, there exists an adaptively secure, collusion-resistant, fully succinct public-key functional encryption scheme. (By "fully succinct", we mean that the runtime of $\mathsf{Enc}(\mathsf{PK}, \mathbf{x})$ *will just be* $\mathsf{poly}(\lambda, |\mathbf{x}|)$ *and not depend on the size of the circuits being evaluated on $\mathbf{x}$.) Moreover, if all of the above assumptions are sub-exponentially secure, then there exists sub-exponentially secure $i\mathcal{O}$ for all circuits.*

Proof. For the sub-exponential security regime, this follows by combining Theorem 5.1, Theorem 6.1, Theorem 6.2, Theorem 7.1, Theorem 7.2, Theorem 7.4, Lemma 7.5, Theorem 7.7, and Theorem 7.9.

In the negligible but not sub-exponential security regime, this follows from instantiating the $\mathsf{SparseLPN}$ assumption with the explicit distribution of [AK23]. As such, we can circumvent the combiners, and we can invoke Theorem 5.1, Theorem 6.1, Theorem 6.2, Theorem 7.1, Theorem 7.2, Theorem 7.4, and lastly [GS16, LM16, ABSV15, KNTY19] to go from selectively secure single-key public-key FE to adaptively secure collusion-resistant public-key FE with polynomial loss. □

Corollary 7.11. *Suppose that the following are true:*

- *The* $\mathsf{SparseLPN}$ *assumption over $\mathbb{Z}_2$;*

- *The* LPN *assumption over large fields; and*

- *The* DLIN *assumption over prime order symmetric bilinear groups.*

Then, there exists an adaptively secure, collusion-resistant, fully succinct public-key functional encryption scheme. Moreover, if all of the above assumptions are sub-exponentially secure, then there exists (sub-exponentially secure) $i\mathcal{O}$ for all circuits.

Proof. This follows from Corollary 3.5, Theorem 4.1, and Corollary 7.10. □

Acknowledgements. We thank Rachel Lin for answering our questions about the constructions by [JLS21, JLS22]. The first author was supported by an Akamai Presidential Fellowship. The second author's research was supported by NSF fellowship DGE-2141064 and by the grants of the third author. The third author's research was supported in part by DARPA under Agreement No. HR00112020023, NSF CNS-2154149, a grant from the MIT-IBM Watson AI, a grant from Analog Devices, a Microsoft Trustworthy AI grant, a Thornton Family Faculty Research Innovation Fellowship from MIT and a Simons Investigator Award. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Government or DARPA.

References

[ABJ+19] Prabhanjan Ananth, Saikrishna Badrinarayanan, Aayush Jain, Nathan Manohar, and Amit Sahai. From FE combiners to secure MPC and back. In Dennis Hofheinz and Alon Rosen, editors, *Theory of Cryptography - 17th International Conference, TCC 2019, Nuremberg, Germany, December 1-5, 2019, Proceedings, Part I*, volume 11891 of *Lecture Notes in Computer Science*, pages 199–228. Springer, 2019. 13, 18, 19, 58, 60

[ABR16] Benny Applebaum, Andrej Bogdanov, and Alon Rosen. A dichotomy for local small-bias generators. *J. Cryptol.*, 29(3):577–596, 2016. 73

[ABSV15] Prabhanjan Ananth, Zvika Brakerski, Gil Segev, and Vinod Vaikuntanathan. From selective to adaptive security in functional encryption. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 657–677. Springer, 2015. 6, 63

[ABW10] Benny Applebaum, Boaz Barak, and Avi Wigderson. Public-key cryptography from different assumptions. In Leonard J. Schulman, editor, *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 171–180. ACM, 2010. 4, 73, 75

[ADI+17] Benny Applebaum, Ivan Damgård, Yuval Ishai, Michael Nielsen, and Lior Zichron. Secure arithmetic computation with constant computational overhead. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 223–254. Springer, 2017. 4

[AIK08] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. On pseudorandom generators with linear stretch in nc^0. *Comput. Complex.*, 17(1):38–69, 2008. 4, 7, 9, 10, 20

[AJ15] Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 308–326. Springer, 2015. 7, 8, 11, 14, 19, 50

[AJS15] Prabhanjan Ananth, Abhishek Jain, and Amit Sahai. Achieving compactness generically: Indistinguishability obfuscation from non-compact functional encryption. *IACR Cryptology ePrint Archive*, 2015:730, 2015. 52

[AK23] Benny Applebaum and Eliran Kachlon. Sampling graphs without forbidden subgraphs and unbalanced expanders with negligible error. *SIAM J. Comput.*, 52(6):1321–1368, 2023. 3, 6, 12, 15, 19, 63, 73, 74

[AKS83] Miklós Ajtai, János Komlós, and Endre Szemerédi. An o(n log n) sorting network. In David S. Johnson, Ronald Fagin, Michael L. Fredman, David Harel, Richard M. Karp, Nancy A. Lynch, Christos H. Papadimitriou, Ronald L. Rivest, Walter L. Ruzzo, and Joel I. Seiferas, editors, *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25-27 April, 1983, Boston, Massachusetts, USA*, pages 1–9. ACM, 1983. 11, 12, 26

[AKV04] Tim Abbot, Daniel Kane, and Paul Valiant. On algorithms for nash equilibria. *Unpublished manuscript*, page 1, 2004. 6

[AL18] Benny Applebaum and Shachar Lovett. Algebraic attacks against random local functions and their countermeasures. *SIAM J. Comput.*, 47(1):52–79, 2018. 4, 73

[Ale11] Michael Alekhnovich. More on average case vs approximation complexity. *Comput. Complex.*, 20(4):755–786, 2011. 4

[AOW15] Sarah R. Allen, Ryan O'Donnell, and David Witmer. How to refute a random CSP. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 689–708. IEEE Computer Society, 2015. 4, 73

[App13] Benny Applebaum. Pseudorandom generators with long stretch and low locality from random local one-way functions. *SIAM J. Comput.*, 42(5):2008–2037, 2013. 73

[AS16] Prabhanjan Vijendra Ananth and Amit Sahai. Functional encryption for turing machines. In Eyal Kushilevitz and Tal Malkin, editors, *Theory of Cryptography - 13th International Conference, TCC 2016-A, Tel Aviv, Israel, January 10-13, 2016, Proceedings, Part I*, volume 9562 of *Lecture Notes in Computer Science*, pages 125–153. Springer, 2016. 6

[BCCD23] Maxime Bombar, Geoffroy Couteau, Alain Couvreur, and Clément Ducros. Correlated pseudorandomness from the hardness of quasi-abelian decoding. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part IV*, volume 14084 of *Lecture Notes in Computer Science*, pages 567–601. Springer, 2023. 73

[BCG$^+$20] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Correlated pseudorandom functions from variable-density LPN. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 1069–1080. IEEE, 2020. 73

[BCG$^+$22] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Nicolas Resch, and Peter Scholl. Correlated pseudorandomness from expand-accumulate codes. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part II*, volume 13508 of *Lecture Notes in Computer Science*, pages 603–633. Springer, 2022. 73

[BCGI18] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 896–912. ACM, 2018. 4

[BGI$^+$12] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2):6:1–6:48, 2012. 3

[BM22] Boaz Barak and Ankur Moitra. Noisy tensor completion via the sum-of-squares hierarchy. *Math. Program.*, 193(2):513–548, 2022. 73

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In Harriet Ortiz, editor, *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 503–513. ACM, 1990. 38

[BNPW20] Nir Bitansky, Ryo Nishimaki, Alain Passelegue, and Daniel Wichs. From cryptomania to obfustopia through secret-key functional encryption. *Journal of Cryptology*, 33(2):357–405, 2020. 14

[BPR15] Nir Bitansky, Omer Paneth, and Alon Rosen. On the cryptographic hardness of finding a nash equilibrium. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 1480–1498. IEEE, 2015. 6

[BQ12] Andrej Bogdanov and Youming Qiao. On the security of goldreich's one-way function. *Comput. Complex.*, 21(1):83–127, 2012. 73

[BS18] Zvika Brakerski and Gil Segev. Function-private functional encryption in the private-key setting. *J. Cryptol.*, 31(1):202–225, 2018. 62

[BV18] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. *J. ACM*, 65(6):39:1–39:37, 2018. 7, 8, 11, 14, 19, 50

[BZ14] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 480–499. Springer, 2014. 6

[CD23] Geoffroy Couteau and Clément Ducros. Pseudorandom correlation functions from variable-density lpn, revisited. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *Public-Key Cryptography - PKC 2023 - 26th IACR International Conference on Practice and Theory of Public-Key Cryptography, Atlanta, GA, USA, May 7-10, 2023, Proceedings, Part II*, volume 13941 of *Lecture Notes in Computer Science*, pages 221–250. Springer, 2023. 73

[CDM$^+$18] Geoffroy Couteau, Aurélien Dupin, Pierrick Méaux, Mélissa Rossi, and Yann Rotella. On the concrete security of goldreich's pseudorandom generator. In Thomas Peyrin and Steven D. Galbraith, editors, *Advances in Cryptology - ASIACRYPT 2018 - 24th*

International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part II, volume 11273 of *Lecture Notes in Computer Science*, pages 96–124. Springer, 2018. 73

[CEMT09] James Cook, Omid Etesami, Rachel Miller, and Luca Trevisan. Goldreich's one-way function candidate and myopic backtracking algorithms. In Omer Reingold, editor, *Theory of Cryptography, 6th Theory of Cryptography Conference, TCC 2009, San Francisco, CA, USA, March 15-17, 2009. Proceedings*, volume 5444 of *Lecture Notes in Computer Science*, pages 521–538. Springer, 2009. 73

[CLP15] Kai-Min Chung, Huijia Lin, and Rafael Pass. Constant-round concurrent zero-knowledge from indistinguishability obfuscation. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 287–307. Springer, 2015. 6

[CLTV15] Ran Canetti, Huijia Lin, Stefano Tessaro, and Vinod Vaikuntanathan. Obfuscation of probabilistic circuits and applications. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II*, volume 9015 of *Lecture Notes in Computer Science*, pages 468–497. Springer, 2015. 6

[CM01] Mary Cryan and Peter Bro Miltersen. On pseudorandom generators in NC. In Jirí Sgall, Ales Pultr, and Petr Kolman, editors, *Mathematical Foundations of Computer Science 2001, 26th International Symposium, MFCS 2001 Marianske Lazne, Czech Republic, August 27-31, 2001, Proceedings*, volume 2136 of *Lecture Notes in Computer Science*, pages 272–284. Springer, 2001. 4, 73

[CPP20] Ran Canetti, Sunoo Park, and Oxana Poburinnaya. Fully deniable interactive encryption. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part I*, volume 12170 of *Lecture Notes in Computer Science*, pages 807–835. Springer, 2020. 6

[CRR21] Geoffroy Couteau, Peter Rindal, and Srinivasan Raghuraman. Silver: Silent VOLE and oblivious transfer from hardness of decoding structured LDPC codes. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part III*, volume 12827 of *Lecture Notes in Computer Science*, pages 502–534. Springer, 2021. 73

[CW14] Jie Chen and Hoeteck Wee. Semi-adaptive attribute-based encryption and improved delegation for boolean formula. In Michel Abdalla and Roberto De Prisco, editors, *Security and Cryptography for Networks - 9th International Conference, SCN 2014, Amalfi, Italy, September 3-5, 2014. Proceedings*, volume 8642 of *Lecture Notes in Computer Science*, pages 277–297. Springer, 2014. 50

[DIJL23] Quang Dao, Yuval Ishai, Aayush Jain, and Huijia Lin. Multi-party homomorphic secret sharing and sublinear MPC from sparse LPN. In Helena Handschuh and Anna

Lysyanskaya, editors, *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part II*, volume 14082 of *Lecture Notes in Computer Science*, pages 315–348. Springer, 2023. 4, 73

[DJ24] Quang Dao and Aayush Jain. Lossy cryptography from code-based assumptions. *IACR Cryptol. ePrint Arch.*, page 175, 2024. 4, 73, 74

[Fei02] Uriel Feige. Relations between average case complexity and approximation complexity. In John H. Reif, editor, *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 534–543. ACM, 2002. 4

[GGH$^+$13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 40–49. IEEE Computer Society, 2013. 3

[GIS18] Sanjam Garg, Yuval Ishai, and Akshayaram Srinivasan. Two-round MPC: information-theoretic and black-box. In Amos Beimel and Stefan Dziembowski, editors, *Theory of Cryptography - 16th International Conference, TCC 2018, Panaji, India, November 11-14, 2018, Proceedings, Part I*, volume 11239 of *Lecture Notes in Computer Science*, pages 123–151. Springer, 2018. 14

[GJLS21] Romain Gay, Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from simple-to-state hard problems: New assumptions, new techniques, and simplification. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part III*, volume 12698 of *Lecture Notes in Computer Science*, pages 97–126. Springer, 2021. 8, 50

[GKP$^+$13] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 555–564. ACM, 2013. 5, 7, 8, 11, 22

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred V. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 218–229. ACM, 1987. 14, 60

[Gol00] Oded Goldreich. Candidate one-way functions based on expander graphs. *IACR Cryptol. ePrint Arch.*, page 63, 2000. 4, 19, 73

[GPS16] Sanjam Garg, Omkant Pandey, and Akshayaram Srinivasan. Revisiting the cryptographic hardness of finding a nash equilibrium. In Matthew Robshaw and Jonathan

Katz, editors, *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 579–604. Springer, 2016. 6

[GS16] Sanjam Garg and Akshayaram Srinivasan. Single-key to multi-key functional encryption with polynomial loss. In Martin Hirt and Adam D. Smith, editors, *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part II*, volume 9986 of *Lecture Notes in Computer Science*, pages 419–442, 2016. 6, 63

[GS22] Sanjam Garg and Akshayaram Srinivasan. Two-round multiparty secure computation from minimal assumptions. *J. ACM*, 69(5):36:1–36:30, 2022. 14, 59

[GVW15] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Predicate encryption for circuits from LWE. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 503–523. Springer, 2015. 8

[HY20] Pavel Hubácek and Eylon Yogev. Hardness of continuous local search: Query complexity and cryptographic lower bounds. *SIAM J. Comput.*, 49(6):1128–1172, 2020. 6

[IKOS08] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Cryptography with constant computational overhead. In Cynthia Dwork, editor, *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 433–442. ACM, 2008. 4

[JLMS19] Aayush Jain, Huijia Lin, Christian Matt, and Amit Sahai. How to leverage hardness of constant-degree expanding polynomials over r to build io. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part I*, volume 11476 of *Lecture Notes in Computer Science*, pages 251–281. Springer, 2019. 50

[JLS19] Aayush Jain, Huijia Lin, and Amit Sahai. Simplifying constructions and assumptions for io. *IACR Cryptol. ePrint Arch.*, page 1252, 2019. 8

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2021, page 60–73, New York, NY, USA, 2021. Association for Computing Machinery. 3, 5, 7, 8, 11, 21, 22, 23, 63, 73

[JLS22] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from LPN over $\mathbb {F}_p$, dlin, and prgs in nc^0. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology - EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 - June 3, 2022, Proceedings, Part I*, volume 13275 of *Lecture*

Notes in Computer Science, pages 670–699. Springer, 2022. 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 17, 18, 19, 21, 22, 23, 24, 26, 32, 33, 40, 41, 42, 43, 44, 45, 50, 52, 63, 73

[JMS20] Aayush Jain, Nathan Manohar, and Amit Sahai. Combiners for functional encryption, unconditionally. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 141–168. Springer, 2020. 6, 10, 13, 14, 15, 18, 19, 33, 44, 56, 58, 59, 60, 61

[KMOW17] Pravesh K. Kothari, Ryuhei Mori, Ryan O'Donnell, and David Witmer. Sum of squares lower bounds for refuting any CSP. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 132–145. ACM, 2017. 4, 73

[KNT17] Fuyuki Kitagawa, Ryo Nishimaki, and Keisuke Tanaka. Indistinguishability obfuscation for all circuits from secret-key functional encryption. *IACR Cryptol. ePrint Arch.*, page 361, 2017. 10, 11, 14, 50

[KNT22] Fuyuki Kitagawa, Ryo Nishimaki, and Keisuke Tanaka. Obfustopia built on secret-key functional encryption. *J. Cryptol.*, 35(3):19, 2022. 6, 8, 10, 11, 14, 15, 19, 44, 50, 58, 62

[KNTY19] Fuyuki Kitagawa, Ryo Nishimaki, Keisuke Tanaka, and Takashi Yamakawa. Adaptively secure and succinct functional encryption: Improving security and efficiency, simultaneously. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 521–551. Springer, 2019. 6, 63

[KNY17] Ilan Komargodski, Moni Naor, and Eylon Yogev. Secret-sharing for NP. *J. Cryptol.*, 30(2):444–469, 2017. 6

[KRS15] Dakshita Khurana, Vanishree Rao, and Amit Sahai. Multi-party key exchange for unbounded parties from indistinguishability obfuscation. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part I*, volume 9452 of *Lecture Notes in Computer Science*, pages 52–75. Springer, 2015. 6

[KS20] Ilan Komargodski and Gil Segev. From minicrypt to obfustopia via private-key functional encryption. *J. Cryptol.*, 33(2):406–458, 2020. 6

[LM16] Baiyu Li and Daniele Micciancio. Compactness vs collusion resistance in functional encryption. In Martin Hirt and Adam D. Smith, editors, *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part II*, volume 9986 of *Lecture Notes in Computer Science*, pages 443–468, 2016. 6, 63

[LP04] Yehuda Lindell and Benny Pinkas. A proof of yao's protocol for secure two-party computation. *IACR Cryptol. ePrint Arch.*, page 175, 2004. 38

[LPST16] Huijia Lin, Rafael Pass, Karn Seth, and Sidharth Telang. Output-compressing randomized encodings and applications. In Eyal Kushilevitz and Tal Malkin, editors, *Theory of Cryptography - 13th International Conference, TCC 2016-A, Tel Aviv, Israel, January 10-13, 2016, Proceedings, Part I*, volume 9562 of *Lecture Notes in Computer Science*, pages 96–124. Springer, 2016. 5, 7, 8, 11, 22

[MST06] Elchanan Mossel, Amir Shpilka, and Luca Trevisan. On epsilon-biased generators in nc^0. *Random Struct. Algorithms*, 29(1):56–81, 2006. 4, 73

[O'D14] Ryan O'Donnell. *Analysis of Boolean Functions.* Cambridge University Press, 2014. 11

[OW14] Ryan O'Donnell and David Witmer. Goldreich's PRG: evidence for near-optimal polynomial stretch. In *IEEE 29th Conference on Computational Complexity, CCC 2014, Vancouver, BC, Canada, June 11-13, 2014*, pages 1–12. IEEE Computer Society, 2014. 73

[PF79] Nicholas Pippenger and Michael J. Fischer. Relations among complexity measures. *J. ACM*, 26(2):361–381, apr 1979. 11, 12, 26

[RRS17] Prasad Raghavendra, Satish Rao, and Tselil Schramm. Strongly refuting random csps below the spectral threshold. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 121–131. ACM, 2017. 73

[RRT23] Srinivasan Raghuraman, Peter Rindal, and Titouan Tanguy. Expand-convolute codes for pseudorandom correlation generators from LPN. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part IV*, volume 14084 of *Lecture Notes in Computer Science*, pages 602–632. Springer, 2023. 73

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 475–484. ACM, 2014. 3, 5, 6

[Wee20] Hoeteck Wee. Functional encryption for quadratic functions from k-lin, revisited. In Rafael Pass and Krzysztof Pietrzak, editors, *Theory of Cryptography - 18th International Conference, TCC 2020, Durham, NC, USA, November 16-19, 2020, Proceedings, Part I*, volume 12550 of *Lecture Notes in Computer Science*, pages 210–228. Springer, 2020. 8, 10, 40, 48, 50, 52

[WW24] Brent Waters and David J. Wu. Adaptively-sound succinct arguments for NP from indistinguishability obfuscation. *IACR Cryptol. ePrint Arch.*, page 165, 2024. 6

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*, pages 162–167. IEEE Computer Society, 1986. 8, 9, 14, 22, 38, 52

A Sparse LPN Cryptanalysis

Many recent works have analyzed variants of LPN over arbitrary finite fields with different constraints on the distributions of the $\mathbf{A}$ matrix [BCG$^+$20, CRR21, BCG$^+$22, CD23, RRT23, BCCD23, DIJL23, DJ24]. Thankfully, prior work has introduced a unified framework to study the security of these variants, called the *linear test framework*. As noted in prior works (see [BCG$^+$20, CRR21, DIJL23]), the existing attacks against sparse LPN over arbitrary fields can be framed as these linear tests. More than that, the linear test framework has been repeatedly used to study Goldreich's PRGs [Gol00, CM01, MST06, CEMT09, BQ12, ABW10, ABR16, App13, OW14, AL18, KMOW17, CDM$^+$18, AK23], used for building polynomial stretch PRGs in NC^0 as is needed in previous constructions of $i\mathcal{O}$ [JLS21, JLS22]. We note that techniques beyond linear tests have been used to attack the special case of sparse LPN over $\mathbb{Z}_2$ [AOW15, BM22, RRS17].

A.1 Linear Test Framework

For simplicity, we restrict attention here to the case of sparse LPN over $\mathbb{Z}_q$ for prime q (which is hence also a finite field). In the linear test framework, we assume that the (computationally unbounded) adversary takes as input the matrix $\mathbf{A} \in \mathbb{F}^{m\times n}$ and outputs some non-zero vector $\mathbf{v} \in \mathbb{F}^m$ such that $\mathbf{v}^\top(\mathbf{As} + \mathbf{e})$ is a *biased* binary random variable over the distribution of $\mathbf{s}$ and $\mathbf{e}$. Notice that this would be sufficient to distinguish LPN. For more formal definitions, see [CRR21, DIJL23, DJ24].

Previous work shows that one can bound the bias in terms of the *dual distance* of the matrix $\mathbf{A}$.

Definition 41 (Dual Distance)**.** *For a finite field $\mathbb{F}$ and a matrix $\mathbf{A} \in \mathbb{F}^{m\times n}$ with $m > n$, we define the* dual distance *of $\mathbf{A}$, denoted $\mathsf{dd}(\mathbf{A})$, to be the minimum Hamming weight of a non-zero vector $\mathbf{v} \in \mathbb{F}^m$ in the left kernel of $\mathbf{A}$.*

Lemma A.1 (Informal, special case of [CRR21])**.** *Suppose the noise rate of the error is η, i.e., the error has distribution $\mathbf{e} \sim \mathsf{Bern}(\mathbb{Z}_q, \eta)^m$. Then, in the linear test framework, the maximum possible bias of any non-zero vector $\mathbf{v}$ is at most $e^{-\Omega(\eta\cdot\mathsf{dd}(\mathbf{A}))}$.*

Now, we refer to previous work to bound $\mathsf{dd}(\mathbf{A})$ for $\mathbf{A} \leftarrow \mathsf{SparseMat}_q(t, n, m)$. For this result, the number of samples is given by $m = O(n^{1+(\frac{t}{2}-1)(1-\alpha)})$ for some constant $\alpha \in (0, 1)$. We note that for $q = 2$, if $m \gg n^{t/2}$, polynomial-time refutation algorithms based on convex programming are known [AOW15, BM22], and for $m \gg n^{1+(\frac{t}{2}-1)(1-\alpha)}$, refutations are known that run in time $\exp(\widetilde{O}(n^\alpha))$ [RRS17].

Lemma A.2 (Lemma A.1 of [DIJL23], Lemma 7.1 of [DJ24])**.** *There exists a universal constant C such that for any field $\mathbb{Z}_q$, $t \geq 3$, $\alpha \in (0, 1)$, and for $m = O(n^{1+(\frac{t}{2}-1)(1-\alpha)})$, the following holds for sufficiently large n:*

$$\Pr\left[\mathsf{dd}(\mathbf{A}) \leq \frac{C}{t}\cdot n^\alpha \;\middle|\; \mathbf{A} \leftarrow \mathsf{SparseMat}_q(t, n, m)\right] = \Theta\left(\left(\frac{t}{n^\alpha}\right)^{t-2}\right).$$

In particular, with at least $1 - 1/\mathsf{poly}(n)$ probability over $\mathbf{A} \leftarrow \mathsf{SparseMat}_q(t, n, m)$, we have $\mathsf{dd}(\mathbf{A}) \geq \Omega(n^\alpha/t)$. Now, we can combine the two above results as follows.

Corollary A.3. *Consider sparse LPN over $\mathbb{Z}_q$ with prime q with constant sparsity t, $m = O(n^{1+(\frac{t}{2}-1)(1-\alpha)})$ for some constant $\alpha \in (0, 1)$, and noise rate $\eta = n^{-\alpha+\beta}$ for some constant $\beta \in (0, \alpha)$. Then, in* *the linear test framework, with probability at least* $1 - 1/\mathsf{poly}(n)$ *over* $\mathbf{A} \leftarrow \mathsf{SparseMat}_q(t, n, m)$*, the maximum possible bias is* $e^{-\Omega(n^\beta/t)}$.

Proof. This directly follows from combining Lemmas A.1 and A.2. □

As a result, setting $\mathsf{GoodSparseMat}_q(t, n, m)$ to be the uniform distribution over

$$\{\mathbf{A} \in \mathsf{SparseMat}_q(t, n, m) : \mathsf{dd}(\mathbf{A}) \geq \Omega(n^\alpha/t)\}$$

and $\mathsf{BadSparseMat}_q(t, n, m)$ to be the uniform distribution over the complement $\mathsf{SparseMat}_q(t, n, m) \setminus \mathsf{GoodSparseMat}_q(t, n, m)$ gives an explicit candidate for the sub-exponential sparse LPN assumption (Definition 7) over $\mathbb{Z}_q$. In particular, it is sub-exponentially secure in the linear test framework.

Remark. Dao and Jain [DJ24] give an attack on sparse LPN in the "compression regime", as would be needed in their applications to construct lossy trapdoor functions [DJ24, Theorem 7.1]. This regime requires an error rate polynomially more aggressive than what is needed to beat the dual distance bound (i.e., as in Lemma A.1 and Corollary A.3). In particular, the attack is a linear test, and the linear test framework does not give a meaningful bound on the bias. For our parameter regime (as in Definition 7 and Corollary A.3), the linear test framework *does* give a meaningful bound on the bias, so in particular, our assumption is not broken by the attack of Dao and Jain [DJ24].

A.2 Alternative Formulations of the Assumption

We briefly mention that we could have chosen stronger variants of the sparse LPN assumption we give (Definition 7) to simplify our analysis. Below are two such possibilities:

- We could have postulated the existence of an *efficiently sampleable* distribution $\mathsf{GoodSparseMat}_q(t, n, m)$ that is plausibly sub-exponentially secure. This algorithm must an output an $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ that does not have a sparse linear combination of rows summing to 0, which is achievable by requiring the t-regular bipartite graph formed with the rows of $\mathbf{A}$ to satisfy certain expansion conditions. This would dramatically simplify our construction, ignoring the need for secret-key FE, FE combiners, and subtleties in our security definitions. Unfortunately, no such efficiently sampleable distribution is known. As such, we choose to work with the weaker assumption given in Definition 7.

  Applebaum and Kachlon [AK23] show that there exists an efficiently sampleable distribution that has bad dual distance with negligible, but not sub-exponential, probability.[6] For polynomially secure public-key functional encryption (i.e., without bootstrapping to $i\mathcal{O}$), such a distribution suffices, and there is no need to consider FE combiners or secret-key FE.

- Alternatively, one could conjecture that for any *fixed* matrix $\mathbf{A}$ with large dual distance, the resulting sparse LPN assumption (with $\mathbf{A}$ fixed) is sub-exponentially secure. This would directly give a non-uniform construction of $i\mathcal{O}$, without needing to consider secret-key FE or FE combiners. This formulation is sufficient to rule out linear tests via the linear test

[6]Explicitly, the stated bound is $n^{-O\left(\frac{\log\log\log n}{\log\log\log\log n}\right)}$ for some parameter range [AK23, Theorem 7.18], [DJ24, Theorem 4.1].

framework, but there is now the possibility that $\mathbf{A}$ has a "trapdoor", in analogy to LWE, that a non-uniform adversary could hardwire to distinguish sparse LPN. We view this as a qualitatively stronger assumption, and as such, we choose to work with the weaker assumption in Definition 7, where there is a lot of entropy in $\mathbf{A}$.

A.3 Public-Key Encryption from Sparse LPN

We briefly mention that the work of Applebaum, Barak, and Wigderson [ABW10] constructs public-key encryption from sparse LPN for constant sparsity $t \geq 3$. However, for decryption, their construction requires that there exists a sufficiently sparse vector in the left kernel of $\mathbf{A}$ (as a secret key) to kill the sparse noise $\mathbf{e}$. As such, their construction needs to be in a parameter regime where the linear test framework *fails*, i.e., the dual distance of $\mathbf{A}$ is not sufficiently large. However, for our parameter regime (as in Definition 7 and Corollary A.3), the linear test framework *does* give a meaningful bound on the bias, so in particular, it is not known how to build public-key encryption from sparse LPN in our parameter regime.