Breaking the Circuit Size Barrier for Secure Computation under Quasi-Polynomial LPN ⋆

Geoffroy Couteau[1] and Pierre Meyer[2]

[1] CNRS, IRIF, Université de Paris, France. couteau@irif.fr
[2] IRIF, Université de Paris, France and IDC Herzliya, Israel. pierre.meyer@ens-lyon.fr

Abstract. In this work we introduce a new (circuit-dependent) *homomorphic secret sharing* (HSS) scheme for any $\log / \log\log$-local circuit, with communication proportional only to the width of the circuit and polynomial computation, which is secure assuming the super-polynomial hardness of *learning parity with noise* (LPN). At the heart of our new construction is a *pseudorandom correlation generator* (PCG) which allows two parties to locally stretch short seeds into pseudorandom instances of an arbitrary $\log / \log\log$-local additive correlation.
Our main application, and the motivation behind this work, is a generic two-party secure computation protocol for every layered (boolean or arithmetic) circuit of size s with total communication $O(s/\log\log s)$ and polynomial computation, assuming the super-polynomial hardness of the standard learning parity with noise assumption (a circuit is layered if its nodes can be partitioned in layers, such that any wire connects adjacent layers). This expands the set of assumptions under which the 'circuit-size barrier' can be broken, for a large class of circuits. The strength of the underlying assumption is tied to the sublinearity factor: we achieve communication $O(s/k(s))$ under the $s^{2^{k(s)}}$-hardness of LPN, for any $k(s) \leq (\log\log s)/4$.
Previously, the set of assumptions known to imply a PCG for correlations of degree $\omega(1)$ or generic secure computation protocols with sublinear communication was restricted to LWE, DDH, and a circularly secure variant of DCR.

Keywords: homomorphic secret sharing · multiparty computation · sublinear communication · learning parity with noise · pseudorandom correlation generators

1 Introduction

In this work, we present a novel (circuit dependent) *homomorphic secret sharing* (HSS) scheme for any ($\log / \log\log$)-local circuit which is secure under the super-polynomial hardness of the learning parity with noise (LPN) assumption. The main application, and motivation for this work, is a new protocol for securely computing layered arithmetic and boolean circuits with communication sublinear in the circuit size, under the quasi-polynomial hardness of LPN.

*Homomorphic Secret Sharing (*HSS*).* An HSS is a compact secret sharing scheme equipped with homomorphism: the parties can locally convert compact shares of an input into (additive) shares of some function of it, without interaction. Compactness here means that the input shares should be much smaller than, and ideally independent of, the size of the evaluated circuit. More precisely, HSS for a circuit class allows the parties to homomorphically convert their shares for any circuit in the class. This powerful primitive has been instantiated for all circuits under LWE [BKS19], or for NC^1 under DDH [BGI16a] or DCR [FGJS17, OSY21, RS21], and for the class of constant degree polynomials from LPN [BCG+19b].

The circuit size barrier in secure computation. Secure computation allows mutually distrustful parties to securely compute a public function of their joint private inputs, concealing all information beyond the output. Since its introduction in the seminal works of Yao [Yao86], and Goldreich, Micali, and Wigderson [GMW87b, GMW87a], secure computation has received a constant attention.

⋆ An extended abstract of this work appeared in Advances in Cryptology – EUROCRYPT 2021, 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques. G. Couteau's research is supported by ANR Project SCENE. This work was done while P. Meyer was a student of the École Normale Supérieure de Lyon, visiting the IRIF and the FACT Center. P. Meyer's research is supported in part by ISF grant 1861/16, AFOSR Award FA9550-17-1-0069, and ERC Starting Grant 852952(HSS).

For a long time, however, all standard approaches to secure computation have been stuck at an intriguing *circuit-size barrier*, in that they require an amount of communication (at least) proportional to the size of the circuit being computed. In contrast, insecure computation only requires exchanging the inputs, which are usually considerably smaller than the entire circuit. Getting beyond this limitation has been a major challenge in secure computation. Early positive results required exponential computation [BFKR91, NN01], or were limited to very simple functions such as point functions [CGKS95, KO97, CG97] or constant-depth circuits [BI05].

The situation changed with the breakthrough result of Gentry [Gen09] on fully-homomorphic encryption (FHE), which led to optimal communication protocols in the computational setting [DFH12, AJL+12]. On the downside, the set of assumptions under which we know how to build FHE is very narrow; it is restricted to lattice-based assumptions such as LWE, and in particular does not include any of the traditional assumptions which were used in the 20th century. More recently, the elegant work of [BGI16a] showed for the first time that secure computation with sublinear communication could be based on assumptions not known to imply FHE, by building a two-party secure computation protocol under the DDH assumption, with communication $O(s/\log s)$ for *layered* circuits of size s.[3] [FGJS17, OSY21, RS21] later followed this blueprint and switched out the DDH assumption for DCR assumption. It remains open whether secure computation with sublinear communication can be based on any other traditional and well-studied assumption, such as code-based assumptions.

1.1 Our Contribution

We show that circuit-dependent homomorphic secret sharing, *i.e.* HSS where the share generation requires knowing in advance the circuit to be evaluated homomorphically, for the class of log-local circuits exists, conditioned on (the quasi-polynomial hardness of) a well-studied 20th century assumption: the learning parity with noise (LPN) assumption [BFKL94]. Informally, the LPN assumption captures the hardness of solving an overdetermined system of linear equations over $\mathbb{F}_2$, when a small subset of the equations is perturbed with a random noise. The LPN assumption has a long history in computational learning theory, where it emerged. Furthermore, our results only require a flavour of LPN where the adversary is given a very limited number of samples (typically, $O(n)$ equations in n indeterminates). In this regime, LPN is equivalent to the hardness of decoding random linear codes over $\mathbb{F}_2$, which is the well-known *syndrome decoding* problem in the coding theory community, where it has been studied since the 60's [Pra62].

Details on the underlying assumption. In a bit more detail, given a security parameter λ, the (T, n, N, r)-LPN assumption with dimension $n = n(\lambda)$, number of samples $N = N(\lambda)$ and noise rate $r = r(\lambda)$ states that for every adversary Adv running in time at most $T = T(\lambda)$,

$$\Pr\left[A \xleftarrow{\$} \mathbb{F}_2^{N\times n}, \boldsymbol{e} \xleftarrow{\$} \mathsf{Ber}_r^N, \boldsymbol{s} \xleftarrow{\$} \mathbb{F}_2^n \ :\ \mathsf{Adv}(A, A\cdot \boldsymbol{s} + \boldsymbol{e}) = \boldsymbol{s}\right] = \mathrm{negl}(\lambda),$$

where Ber_r denotes the Bernouilli distribution which outputs 1 with probability r, and negl denote some negligible function. When T can be any polynomial (resp. any super-polynomial function, some super-polynomial function), we say that we assume the polynomial (resp. quasi-polynomial, super-polynomial) hardness of LPN. For arithmetic circuits, we need to assume LPN over large fields, or equivalently syndrome decoding for random linear codes over large fields; this is also a well-founded and well-studied assumption, used in several previous works, e.g. [BCGI18, BCG+19b].

HSS for Any loglog-Depth Circuit. We introduce a new circuit-dependent HSS scheme for the class of any log log-depth circuits. We emphasise that unlike traditional forms of HSS, here the input-sharing phase depends on the homomorphic evaluation circuit: in other words it is an HSS scheme for any singleton class comprised of a single circuit of depth log log, not for the class of all such circuits simultaneously.

Main Theorem 1 (HSS for any loglog-Depth Circuit, Informal)**.** *Let C be a size-s, n-input, m-output, $(\epsilon \cdot \log\log)$-depth arithmetic circuit over $\mathbb{F}$ (for some $\epsilon \leq 1/4$). If the $\mathbb{F}$-LPN assumption with super-polynomial dimension ℓ, $O(\ell)$ samples, and inverse super-polynomial rate holds then there exists a secure HSS scheme for the class $\{C\}$ with share size $n + O(m \cdot s \cdot \log s / c^{\log^{1-\epsilon} s - \log^{1-2\epsilon} s})$ (for some constant c) and computational complexity $O(m \cdot \mathsf{poly}(s) \cdot (\log |\mathbb{F}|)^2)$.*

[3] A depth-d circuit is layered if it can be divided into d layers such that any wire connects adjacent layers.

Restricting the circuit class to depth-k size-s circuits where $k(s) \leq \log\log s/4$ leads to quantitative improvements in the size of the shares, the computational complexity of expanding shares, and the strength of the LPN assumption.

Application to Sublinear Computation. Our HSS scheme has (non black-box) implications for sublinear computation. As in [BGI16a], our results holds for all layered (boolean or arithmetic) circuits, in the two-party setting.

Main Theorem 2 (Sublinear Computation of Layered Circuits, Informal)**.** *For any layered arithmetic circuit C of polynomial size $s = s(\lambda)$ with n inputs and m outputs, for any function $k(s) \leq \log\log s - \log\log\log s + O(1)$, there exists a two party protocol for securely computing C in the honest-but-curious model, with total communication $2(n+m+s/k)\cdot\log|\mathbb{F}| + o(s/k)$ and computation bounded by $s^3 \cdot \mathsf{polylog}(s)\cdot(\log|\mathbb{F}|)^2$ under a set of LPN assumptions, the exact nature of which depends on the sublinearity factor k.*

In particular, setting $k \leftarrow O(\log\log s)$ leads to a protocol with total communication $O(n + m + s/\log\log s)$, secure under the super-polynomial hardness of:

- $\mathbb{F}$-LPN *with super-polynomial dimension ℓ, $O(\ell)$ samples, and inverse super-polynomial rate,*
- $\mathbb{F}_2$-LPN *with super-polynomial dimension ℓ', $O(\ell')$ samples, and inverse polynomial rate $1/s^{O(1)}$ (which is implied by the above if $\mathbb{F} = \mathbb{F}_2$).*

Furthermore (but with a slighly different choice of parameters than the one described above), as k is reduced to an arbitrarily small $k = \omega(1)$, we need only assume the quasi-polynomial hardness of:

- $\mathbb{F}$-LPN *with quasi-polynomial dimension ℓ, $O(\ell)$ samples, and inverse quasi-polynomial rate,*
- $\mathbb{F}_2$-LPN *with quasi-polynomial dimension ℓ', $O(\ell')$ samples, and inverse polynomial rate $1/s^{O(1)}$ (which is implied by the above if $\mathbb{F} = \mathbb{F}_2$).*

and the computation is reduced to $O(s^{1+o(1)} \cdot (\log|\mathbb{F}|)^2)$.

Remark 1. While we require security against super-polynomial-time adversaries, this remains a relatively weak flavour of LPN where the dimension is very high, i.e. super-polynomial as well (and the adversary is allowed to run in time $O(\ell^2)$ where ℓ is the dimension), and the number of samples which the adversary gets is very limited, $O(\ell)$. On the other hand, we require a very small noise rate λ/N. For example, instantiating the above with $k = (\log\log s)/5$, we obtain a secure computation protocol with total communication $O(\ell + m + s/\log\log s)$ (sublinear in s) and polynomial computation, assuming that LPN is hard against adversaries running in super-polynomial time $\lambda^{O(\log\lambda)}$, with dimension $\ell = \lambda^{O(\log\lambda)}$, $N = 2\ell$ samples, and noise rate λ/N. More generally, for any super-constant function $\omega(1)$, there is a two-party protocol with communication $O(n+m+s/\log\omega(1))$ assuming the $\lambda^{\omega(1)}$-hardness of LPN (i.e., the quasi-polynomial hardness of LPN).

We note that, in this regime of parameters, the best known attacks are the information set decoding attack [Pra62] and its variants (which only shave constant in the exponents, hence have the same asymptotic complexity), which require time $2^{O(\lambda)}$.[4] Therefore, assuming hardness against $\lambda^{O(\log\lambda)}$-time adversaries is a very plausible assumption.

Remark 2 (On the Generality of Layered Circuits). Our construction is restricted to the class of (boolean or arithmetic) layered circuits. This restriction stems from the blockwise structure of the construction, and was also present in the previous works of [BGI16a] and [Cou19]. As noted in [Cou19], layered circuits are a relatively large and general class of circuits, which furthermore capture many "real-world" circuits such as FFT-like circuits (used in signal processing, integer multiplication, or permutation networks [Wak68]), Symmetric crypto primitives (e.g. AES and algorithms that proceed in sequences of low-complexity rounds are naturally "layered by blocks"), or dynamic-programming algorithm (e.g. the Smith-Waterman distance, or the Levenshtein distance and its variants).

[4] BKW and its variants [BKW00,Lyu05] do not improve over information set decoding attacks in this regime of parameters, due to the very low number of samples.

Generalisation to the malicious setting. Our result can directly be generalised to the malicious setting using a generic GMW-style compiler [GMW87a], which is communication preserving when instantiated with succinct zero-knowledge arguments [NN01]. Such arguments exist under collision-resistant hash functions; hence, Theorem 2 extends to the malicious setting as well, at the cost of further assuming collision-resistant hash functions (which is a mild assumption). We note that CRHFs have recently been built from (sub-exponentially strong) flavours of LPN [AHI+17, YZW+19, BLVW19].

1.2 Our Techniques

Our starting point is the construction of *pseudorandom generator* (PCG) from the work of [BCG+19b], under the LPN assumption. At a high level, a PCG allows to distributively generate long pseudorandom instances of a correlation. More precisely, a PCG for a correlation corr (seen as a distribution over pairs of elements) is a pair $(\mathsf{Gen}, \mathsf{Expand})$ where $\mathsf{Gen}(1^\lambda)$ generates a pair of seeds $(\mathsf{k}_0, \mathsf{k}_1)$ and $\mathsf{Expand}(b, \mathsf{k}_b)$ output a string R_b. A PCG must satisfy two properties: (correctness) (R_0, R_1) is indistinguishable from a random sample from corr, and (security) for $b \in \{0,1\}$, the string R_b is indistinguishable, even given k_{1-b}, from a string R'_b sampled randomly conditioned on satisfying the correlation with R_{1-b}.

The technical contribution at the heart of this paper is to show that, under a certain LPN assumption, there exists a 2-party PCG for the following correlation, which we call *substrings tensor powers* (stp) correlation. It is (publicly) parametrised by

– a string length n;
– subsets $S_1, \ldots, S_{\mathsf{n_s}} \in \binom{[n]}{\leq K}$ of at most $K = \log n / \log\log n$ many coordinates each;
– a tensor power parameter tpp (which can be super-constant, as high as K);

and generates additive shares of all the tensor powers of the prescribed substrings of a random string, *i.e.*

$$(\boldsymbol{r}, ((1_\mathbb{F} \mid\mid \boldsymbol{r}[S_i])^{\otimes \mathsf{tpp}})_{1 \leq i \leq \mathsf{n_s}}), \text{ where } \boldsymbol{r} \in \mathbb{F}^n \text{ is (pseudo)random.}$$

In the above, $\boldsymbol{a}^{\otimes b}$ denotes a vector $\boldsymbol{a}$ tensored with itself b rimes. In order to build shares of $(\boldsymbol{r}, \boldsymbol{r}^{\otimes 2})$ for some (pseudo)random $\boldsymbol{r} \in \mathbb{F}^n$ (the bilinear correlation), the PCG of [BCG+19b] uses a multi-point function secret sharing scheme (MPFSS) (defined in section 3.1) to give the parties small seeds which can be expanded locally to shares of $(\boldsymbol{e}, \boldsymbol{e}^{\otimes 2})$ for some random sparse vector $\boldsymbol{e} \in \mathbb{F}^n$. Thence, if H is some suitable public matrix the parties can get shares of $\boldsymbol{r} := H \cdot \boldsymbol{e}$, which is pseudorandom under LPN, and of $\boldsymbol{r}^{\otimes 2} = H^{\otimes 2} \cdot \boldsymbol{e}^{\otimes 2}$ by locally multiplying their shares of $\boldsymbol{e}$ and $\boldsymbol{e}^{\otimes 2}$ by H and $H^{\otimes 2}$ respectively. The main issue in using this approach directly is that performing the expanding $\boldsymbol{r}^{\otimes \mathsf{tpp}} = H[S_i]^{\otimes \mathsf{tpp}} \cdot \boldsymbol{e}^{\otimes \mathsf{tpp}}$ (where $H[S_i]$–abusively–denotes the submatrix of H with only the rows indexed by elements of S_i) would require super-polynomial computation, as $H[S_i]$ has n columns.

The core idea of our work is to develop a very careful modified strategy. Instead of letting each $\boldsymbol{r}$ be a (pseudo)random mask, we construct $\boldsymbol{r}$ as a sum of $n \cdot \log n$ vectors $\boldsymbol{r}_j$, each associated with a public subset of at most K coordinates: these K coordinates are random, but all others are zero. The crucial property achieved by this construction is the following: with high probability, the sum of these sparse vectors will be pseudorandom, but every size-K substring of $\boldsymbol{r}$ (and in particular $S_1, \ldots, S_{\mathsf{n_s}}$) will be expressible as a sum of 'not too many' of the $\boldsymbol{r}_j$. This allows the expanding to be done by raising to the tensor power tpp a matrix whose dimensions are both $K^{O(1)}$, and not n as before. Thus computation remains polynomial.

If we were to stop here, the size of the seeds would grow linearly with $\mathsf{n_s}$, the number of subsets; this would violate the compactness requirement. Instead, we show that we can batch the subsets into $\mathsf{n_s}/\beta$ groups of at most β subsets each, for some parameter β to be refined, to reduce the share size and recover compactness, without harming computational efficiency. Indeed, so long as β is not too large, the substring of $\boldsymbol{r}$ associated with the union of any β size-K subsets of coordinates will still be expressible as a sum of 'not too many' of the $\boldsymbol{r}_j$. Our computations reveal a sweet spot for the choice of β, for which the PCG seeds are compact and yet the complexity of expanding them remains polynomial.

1.3 Related Work

Pseudorandom correlation generators were first studied (under the name of cryptocapsules) in [BCG$^+$17]. Constructions of PCGs for various correlations, under variants of the LPN assumptions, and applications of PCGs to low-communication secure computation, have been described in [BCGI18, BCG$^+$19b, BCG$^+$19a, SGRR19, BCG$^+$20b, BCG$^+$20a].

Early works on sublinear-communication secure computation either incurred some exponential cost, or were restricted to very limited types of computations. The first protocols to break the circuit size barriers was shown in [BFKR91] (which gave a protocol with optimal communication, albeit with exponential computation and only for a number of parties linear in the input size). The work of [NN01] gave a sublinear protocol, but with exponential complexity. The work of [BI05] gives a low-communication protocol for constant-depth circuit, for a number of parties polylogarithmic in the circuit size, and the works of [CGKS95, KO97, CG97] gave sublinear protocols for the special case of point functions. The result of Gentry [Gen09] led to the first optimal communication protocols in the computational setting [DFH12, AJL$^+$12] under LWE-style assumptions, for all circuits and without incurring any exponential cost. The work of [IKM$^+$13] gave an optimal communication protocol in the correlated randomness model, albeit using an exponential amount of correlated randomness. More recently, [Cou19] constructed an unconditionally secure MPC protocol with sublinear communication for layered circuits, in the two-party setting, with a polynomial amount of correlated randomness. Finally, progress in breaking the circuit-size barrier for layered circuits in the computational setting is closely tied to the advances in HSS for super-constant depth circuits [BGI16a, FGJS17, OSY21, RS21].

2 Technical Overview

Notations. We say that a function negl: $\mathbb{N} \rightarrow \mathbb{R}^+$ is *negligible* if it vanishes faster than every inverse polynomial. For two families of distributions $X = \{X_\lambda\}$ and $Y = \{Y_\lambda\}$ indexed by a security parameter $\lambda \in \mathbb{N}$, we write $X \overset{c}{\approx} Y$ if X and Y are *computationally indistinguishable* (*i.e.* any family of circuits of size $\mathsf{poly}(\lambda)$ has a negligible distinguishing advantage), $X \overset{s}{\approx} Y$ if they are *statistically indistinguishable* (*i.e.* the above holds for arbitrary, unbounded, distinguishers), and $X \equiv Y$ if the two families are identically distributed.

We usually denote matrices with capital letters (A, B, C) and vectors with bold lowercase $(\boldsymbol{x}, \boldsymbol{y})$. By default, vectors are assumed to be column vectors. If $\boldsymbol{x}$ and $\boldsymbol{y}$ are two (column) vectors, we use $\boldsymbol{x}||\boldsymbol{y}$ to denote the (column) vector obtained by their concatenation. We write $\boldsymbol{x} \otimes \boldsymbol{y}$ to denote the tensor product between $\boldsymbol{x}$ and $\boldsymbol{y}$, i.e., the vector of length $n_x n_y$ with coordinates $x_i y_j$ (where n_x is the length of $\boldsymbol{x}$ and n_y is the length of $\boldsymbol{y}$). We write $\boldsymbol{x}^{\otimes 2}$ for $\boldsymbol{x} \otimes \boldsymbol{x}$, and more generally, $\boldsymbol{x}^{\otimes n}$ for the *n-th tensor power* of $\boldsymbol{x}$, $\boldsymbol{x} \otimes \boldsymbol{x} \otimes \cdots \otimes \boldsymbol{x}$. Given a vector $\boldsymbol{x}$ of length $|\boldsymbol{x}| = n$, the notation $\mathsf{HW}(x)$ denotes the Hamming weight $\boldsymbol{x}$, *i.e.* , the number of its nonzero entries. Let k be an integer. We let $\{0,1\}^k$ denote the set of bitstrings of length k. For two strings (x, y) in $\{0,1\}^k$, we denote by $x \oplus y$ their bitwise xor.

Circuits. An arithmetic circuit C with n inputs and m outputs over a field $\mathbb{F}$ is a directed acyclic graph with two types of nodes: the *input nodes* are labelled according to variables $\{x_1, \cdots, x_n\}$; the *(computation) gates* are labelled according to a base B of arithmetic functions. In this work, we will focus on arithmetic circuits with indegree two, over the standard basis $\{+, \times\}$. C contains m gates with no children, which are called *output gates.* If there is a path between two nodes (v, v'), we say that v is an *ancestor* of v'. In this work, we will consider a special type of arithmetic circuits, called *layered arithmetic circuits* (LBC). An LBC is a arithmetic circuit C whose nodes can be partitioned into $D = \mathsf{depth}(C)$ layers $(L_1, \cdots, L_d)$, such that any edge (u, v) of C satisfies $u \in L_i$ and $v \in L_{i+1}$ for some $i \leq d - 1$. Note that the width of a layered arithmetic circuit is also the maximal number of non-output gates contained in any single layer. Evaluating a circuit C on input $\boldsymbol{x} \in \mathbb{F}^n$ is done by assigning the coordinates of $\boldsymbol{x}$ to the variables $\{x_1, \cdots, x_n\}$, and then associating to each gate g of C (seen as an arithmetic function) the value obtained by evaluating g on the values associated to its parent nodes. The output of C on input $\boldsymbol{x}$, denoted $C(\boldsymbol{x})$, is the vector of values associated to the output gates.

2.1 PCG and HSS

Much like a PCG for the bilinear correlation yields an HSS for degree-two circuits [BCG+19b], given a PCG for the stp correlation with $\mathsf{tpp} = K$, it is almost immediate to build an HSS scheme for any singleton class comprised of a log/loglog-local circuit C (which is the case in particular if its depth is at most $\log\log - \log\log\log$, since the gates have in-degree at most 2). Since the circuit to be homomorphically evaluated on the input shares is known, the Share procedure can depend on it (which is not usually the case for HSS). Let $S_1, \ldots, S_m$ be the subsets of inputs on which each output depends, and let K denote the locality of C; we build a (circuit dependent) HSS scheme as follows:

- HSS.Share($\boldsymbol{x}$): Generates compact PCG key (k_0, k_1) which expand to shares of $(\boldsymbol{r}, ((1_{\mathbb{F}} \mid\mid \boldsymbol{r}[S_i])^{\otimes \mathsf{tpp}})_{1\leq i\leq m})$, set $\boldsymbol{x}' \leftarrow \boldsymbol{x} \oplus \boldsymbol{r}$, and give to each party P_σ a share $s_\sigma = (k_\sigma, \boldsymbol{x}')$.
- HSS.Eval(σ, s_σ): Expand s_σ and, for each $i = 1 \ldots m$, extract a share of $(1_{\mathbb{F}} \mid\mid \boldsymbol{r}[S_i])^{\otimes \mathsf{tpp}}$. Use it to generate shares of the coefficients of the "degree-K polynomial" on $|S_i| \leq K$ variables P_i satisfying $P_i(X) = C(X - \boldsymbol{r}[S_i])$. Output the inner product of the vector of coefficient shares with the vector $(1_{\mathbb{F}} \mid\mid \boldsymbol{x}')^{\otimes K}$. (This linear product is a share of $P_i(\boldsymbol{x}')$.)

Correctness and security follow from inspection, along the same lines as [BCG+19b]. Usually, HSS.Share is given only a circuit class as auxiliary input, not a specific circuit, and the parties should be able to homomorphically evaluate any circuit in the class. In our case however the HSS is circuit-dependent, because the subsets $S_1, \ldots, S_m$ are intrinsically tied to the evaluated circuit. An alternative formulation is that our HSS scheme supports singleton circuit classes (or, more generally, local circuits with the same pattern of subsets).

2.2 Generating Correlated Randomness from a PCG

From now on, we set the number of parties to $N = 2$. The work of [BCG+19b, Section 6] provides a pseudorandom correlation generator under the LPN assumption, generates correlated (pseudo) random strings for the low-degree polynomial correlation, *i.e.* shares of $(\boldsymbol{r}, \boldsymbol{r}^{\otimes 2}, \ldots, \boldsymbol{r}^{\otimes d})$ for some constant d, where $\boldsymbol{r}$ is a (pseudo)random vector. With the construction from the previous paragraph, this yields an HSS for constant-depth circuits. Our goal is to design a PCG which would lead to an HSS for super-constant depth circuits. More specifically, and keeping our end application in mind, we would like for our PCG to have short enough seeds to lead to a *compact* HSS scheme (i.e., shares of an input x should be at most $O(x)$). This is fundamental when using the scheme to generate correlated randomness in the protocol of [Cou19], which achieves sublinear communication in the correlated randomness model, and which is the starting point of our application to sublinear secure computation.

Our approach is therefore to directly plug in the construction of [BCG+19b] and see where it fails. Two issues emerge: the computation is super-polynomial, and the communication not sublinear. Below, we outline each of these issues, and explain how we overcome them.

First Issue: Too Many Polynomials. The first problem which appears when plugging the PCG of [BCG+19b] in the protocol of [Cou19] is that the latter requires distributing *many* shares of multivariate polynomials $\hat{Q}$ – more precisely, s/k such polynomials (one for each coordinate of each first layer of a bloc). While the PCG of [BCG+19b] allows to compress pseudorandom pairs $(\boldsymbol{r}, Q(\boldsymbol{X} - \boldsymbol{r}))$ into short seeds, these seeds will still be of length at least $\omega(\log \lambda)$, where λ is the security parameter, for the PCG to have any hope of being secure. That means that even if we could manage to securely distribute all these seeds with optimal communication protocols, the overall communication would still be at the very least $\omega((s \log \lambda)/\log\log s)$, which cannot be sublinear since $\log\log s = o(\log \lambda)$ (as s is polynomial in λ).

We solve this first issue as follows: we fix a parameter β, and partition each $\boldsymbol{y}_i$ into w/β subvectors, each containing β consecutive coordinates of $\boldsymbol{y}_i$. Then, the core observation is that a simple variant of the PCG of [BCG+19b] allows in fact to generate shares of $(\boldsymbol{r}, \boldsymbol{r}^{\otimes 2}, \cdots, \boldsymbol{r}^{\otimes 2^k})$ for some pseudorandom r, where $\boldsymbol{r}^{\otimes j}$ denotes the tensor product of $\boldsymbol{r}$ with itself j times (which we call from now on the *j-th tensor power of* $\boldsymbol{r}$): this correlation is enough to generate shares of all degree-2^k polynomial in $\boldsymbol{r}$ rather than a single one. We will build upon this observation to show how to generate a batch of β shares of multivariate polynomials from a single tensor-power correlation, thus reducing the number of PCG seeds required in the protocol by a factor of β, at the tolerable cost of slightly increasing the size of each seed.

Solution: Batching β Multivariate Polynomials. Consider the first length-β subvector of $\boldsymbol{y}_{i+1}$, which we denote $\boldsymbol{v}$. Observe that the entire subvector $\boldsymbol{v}$ can depend on at most $\beta \cdot 2^k$ coordinates of $\boldsymbol{y}_i$, since each coordinate of $\boldsymbol{v}$ depends on at most 2^k coordinates of $\boldsymbol{y}_i$. Therefore, we can now see the computation of $\boldsymbol{v}$ from $\boldsymbol{y}_i$ as evaluating β multivariate polynomials $(Q_1 \cdots, Q_\beta)$, where all multivariate polynomials take as input the same size-$(\beta 2^k)$ subset of coordinates of $\boldsymbol{y}_i$. To securely compute shares of $\boldsymbol{v}$ from shares of $\boldsymbol{y}_i$, the parties can use the following type of correlated randomness: they will have shares of $(\boldsymbol{r}, \boldsymbol{r}^{\otimes 2}, \cdots \boldsymbol{r}^{\otimes 2^k})$, where $\boldsymbol{r}$ is a random mask of length $\beta \cdot 2^k$. Consider the following polynomials:

$$(\hat{Q}_1(\boldsymbol{X}), \cdots, \hat{Q}_\beta(\boldsymbol{X})) \stackrel{\text{def}}{=} (Q_1(\boldsymbol{X} - \boldsymbol{r}), \cdots, Q_\beta(\boldsymbol{X} - \boldsymbol{r})).$$

Each coefficient of each $\hat{Q}$ can be computed as a degree-2^k multivariate poynomial in the coordinates of $\boldsymbol{r}$ – or, equivalently, as a linear combination of the coordinates of $(\boldsymbol{r}, \boldsymbol{r}^{\otimes 2}, \cdots \boldsymbol{r}^{\otimes 2^k})$. Hence, given additive shares of $(\boldsymbol{r}, \boldsymbol{r}^{\otimes 2}, \cdots \boldsymbol{r}^{\otimes 2^k})$, the parties can locally compute additive shares of the coefficients of *all* the polynomials $(\hat{Q}_1, \cdots \hat{Q}_\beta)$. Using the PCG of [BCG+19b], the seeds for generating pseudorandom correlations of the form $(\boldsymbol{r}, \boldsymbol{r}^{\otimes 2}, \cdots \boldsymbol{r}^{\otimes 2^k})$ have length:

$$O\left(\lambda^{2^k} \cdot \log\left(\left(\beta \cdot 2^k\right)^{2^k}\right)\right),$$

where λ is some security parameter related to the hardness of the underlying LPN assumption. Or more simply, using the fact the computational cost of generating the correlations contains the term $\left(\beta \cdot 2^k\right)^{2^k}$ which must remain polynomial in s. Therefore, the total number of bits which the parties have to distribute (for all $(d/k) \cdot (w/\beta) = s/(\beta k)$ such seeds) is $O((s/k) \cdot (\lambda^{2^k} \cdot \log s)/\beta)$.

Choosing the Parameter β. Suppose for simplicity that we already have at hand an MPC protocol allowing to securely distribute such seeds between the parties, with linear overhead over the total length of the seeds generated. This means that generating the full material will require a total communication of $c \cdot s \cdot \lambda^{2^k} \cdot \log s/(\beta k)$. By setting β to be larger than $c \cdot \lambda^{2^k} \cdot \log s$, the total communication will be upper bounded by $O(s/k) = O(s/\log\log s)$ when setting $k \leftarrow O(\log\log s)$, which is the highest our techniques will allow it to be pushed. The most important remaining question is whether we can execute this process in polynomial time given such a large β. Put more simply, the core issue is that the *computational complexity* of expanding short seeds to shares of $(\boldsymbol{r}, \boldsymbol{r}^{\otimes 2}, \cdots \boldsymbol{r}^{\otimes 2^k})$ with the PCG of [BCG+19b] contains a term of the form $(\beta \cdot 2^k)^{2^k}$. To make the computation polynomial, we must therefore ensure that β is at most $s^{O(2^{-k})}$, which is subpolynomial. Fortunately, this can be done by setting the security parameter λ of the underlying PCG to be $s^{O(2^{-2k})}$. For instance, for any constant $\epsilon \in]0, 1[$, we can set $\lambda \leftarrow 2^{\log^\epsilon s}$, $k \leftarrow \log\log s/c_\epsilon$, and $\beta \leftarrow s^{O(2^{-k})}$ for some explicit constant $c_\epsilon > 2$, at the cost of now having to assume the *quasi-polynomial security* of the LPN assumption.

Second Issue: Too Much Communication. In the previous paragraphs, we focused on generating the appropriate correlated random coins using sublinear total communication. But doing so, we glossed over the fact that in the full protocol, the parties must *also* broadcast (shares of) values of the form $\boldsymbol{y} + \boldsymbol{r}$, where $\boldsymbol{y}$ contains values of some layer, and $\boldsymbol{r}$ is some mask. Recall that with the method which we just outlined, the parties must generate such a length-$(\beta 2^k)$ mask $\boldsymbol{r}$ for the k-ancestors of each length-β subvector of each last layer of a block. Since there are d/k blocks, whose first layers contain w/β subvector each, and since each $\boldsymbol{y} + \boldsymbol{r}$ is of length $\beta \cdot 2^k$, this requires to communicate a total of $(d/k) \cdot (w/\beta) \cdot \beta 2^k = s \cdot 2^k/k$ values – and this cannot possibly be sublinear in s. In fact, this issue already appears in [Cou19], where it was solved as follows: rather than picking an independent mask for each vector of ancestors of a node on a layer (or, in our case, of a length-β block of nodes), pick a single $\boldsymbol{r}_i$ to mask a full layer $\boldsymbol{y}_i$, and define the mask for the subset $S_{i,j}$ of ancestors of a target value $y_{i+1,j}$ to be $\boldsymbol{r}_i[S_{i,j}]$. This implies that the parties must mow broadcast a single masked vector $\boldsymbol{y}_i + \boldsymbol{r}_i$ for each first layer of a block, reducing the overall communication back to $O(s/k)$. The correlated randomness which the parties must securely distribute now consists of tensor powers of many subsets of the coordinates of each mask.

Using the PCG of [BCG$^+$19b] for 'Subvectors Tensor Powers Correlations'. However, attemping to construct a PCG for generating this kind of correlated randomness from the PCG of [BCG$^+$19b] blows up the computation to the point that it can no longer be polynomial. To explain this issue, we briefly recall the high level construction of the PCG of [BCG$^+$19b]. To share a pseudorandom vector $(\boldsymbol{r}, \cdots, \boldsymbol{r}^{\otimes 2^k})$ where $\boldsymbol{r}$ is of length w, the PCG will first generate a *very sparse* vector $\boldsymbol{r}'$, with some number t of nonzero coordinates. Then, each $(\boldsymbol{r}')^{\otimes n}$ for some $n \leq 2^k$ is itself a t^n-sparse vector, of length w^n. Using multi-point function secret sharing (MPFSS, a primitive which was developed in a recent line of work [GI14, BGI15, BGI16b, BCGI18] and can be built from one way functions), one can compress shares of $(\boldsymbol{r}')^{\otimes n}$ to length-$t^n \cdot \log w$ seeds. Then, the final pseudorandom correlation is obtained by letting the parties locally compress $\boldsymbol{r}'$ by multiplying it with a large public matrix H, giving a vector $\boldsymbol{r} = H \cdot \boldsymbol{r}'$. Similarly, $\boldsymbol{r}^{\otimes n}$ can be reconstructed by computing $H^{\otimes n} \cdot (\boldsymbol{r}')^{\otimes n} = (H \cdot \boldsymbol{r}')^{\otimes n} = \boldsymbol{r}^{\otimes n}$, using the multilinearity of tensor powers. The security relies on the fact that if H is a large compressing public random matrix, then its product with a random sparse noise vector $\boldsymbol{r}'$ is indistinguishable from random, under the dual LPN assumption (which is equivalent to the standard LPN assumption). Concretely, one can think of $\boldsymbol{r}'$ as being of length $2w$, and of H as being a matrix from $\mathbb{F}^{w \times 2w}$ which compresses $\boldsymbol{r}'$ to a pseudorandom length-w vector.

Now, the issue with this construction is that even if we need only tensor powers of small subvectors (of length $\beta \cdot 2^k$ in our construction) of the vector $\boldsymbol{r}$, the computation for expanding the seed to these pseudorandom tensor powers will grow super-polynomially with the length of of *entire* vector w. Indeed, consider generating the 2^k-th tensor power of a subvector $\boldsymbol{r}[S]$ of $\boldsymbol{r}$, for some size-$\beta \cdot 2^k$ subset S of $[w]$. Then with the PCG of [BCG$^+$19b], this requires computing $(H[S])^{\otimes 2^k} \cdot (\boldsymbol{r}'[S])^{\otimes 2^k}$, where the share of $(\boldsymbol{r}'[S])^{\otimes 2^k}$ are obtained from a short seed using MPFSS, and $H[S] \in \mathbb{F}^{|S| \times 2w}$ is the submatrix of H whose columns are indexed by S. The core issue becomes now visible: even though $H[S]$ has only $|S|$ rows, it still has $2w$ columns, and computing $H[S]^{\otimes 2^k}$ requires roughly $(|S| \cdot w)^{2^k}$ arithmetic operation. But since we want ultimately to have k be some increasing function of s, the above will contain a term of the form $w^{2^k} = w^{\omega(1)}$, where w (the circuit width) can be polynomial in the circuit size s, leading to an overall computational complexity of $s^{\omega(1)}$, which is super-polynomial.

Solution: Covering the Private Values with the Sum of Separable Masks. Our solution to circumvent the above problem is to generate $\boldsymbol{r}$ as the sum of a certain number m of shorter masks $\boldsymbol{r}^1, \boldsymbol{r}^2, \ldots$ which each only cover θ values (note that they may – and will – overlap). This way the 2^k-th tensor power of a subvector $\boldsymbol{v}$ can be obtained from appropriate linear combinations of coordinates of the 2^k-th tensor power of the concatenation of *only* the $\boldsymbol{r}^j$ which overlap with $\boldsymbol{v}$. The amount of computation grows super-polynomially in the length of this concatenated vector only (instead of w as before).

More formally, we have a list of w/β target subsets $S_1, \ldots, S_{w/\beta}$ (each one corresponding to the $2^k\beta$ ancestors of a batch of β outputs) for which we want to compute the 2^k-th tensor power of $\boldsymbol{r}[S_i]$, for some random $\boldsymbol{r} \in \mathbb{F}^w$. We want to find M size-K sets $\alpha_1, \alpha_2, \ldots, \alpha_M \in \binom{[w]}{K}$ such that each S_i intersects with a small number B of α_js, while $\cup_{i=1}^M \alpha_i = [w]$. We associate each α_j with a vector $\boldsymbol{r}^j \in \mathbb{F}^K$: together they define a sparse subvector of $\mathbb{F}^w$. If we let $\boldsymbol{r}$ be the sum of these sparse vectors, it is clear that for any $i \in [w/\beta]$, each element of $(1_{\mathbb{F}} \mid\mid \boldsymbol{r}[S_i])^{\otimes 2^k}$ can be obtained by a linear combination of the elements of the 2^k-th tensor power of the vector of size $(1 + BK)$ obtained by concatenating $(1_{\mathbb{F}})$ and the $\boldsymbol{r}^j$s such that $\alpha_j \cap S_i \neq \emptyset$. The amount of computation required is then of the order $(BK)^{2^k}$.

The problem of deterministically finding such subsets $\alpha_1, \ldots, \alpha_M$ – which we call a *B-Good Cover* of $(S_i)_{i \in [w/\beta]}$ – turns out to be difficult in the general case. Fortunately, there is a straightforward probabilistic solution: choosing them independently and at random works with high probability. More specifically, taking $M \leftarrow O(w \cdot \ln w)$ i.i.d. uniformly random submasks covering $K \leftarrow \beta 2^k$ values each means that the $\beta 2^k$ ancestral inputs of any batch of β outputs will be covered by only a total of roughly $B = \log w$ submasks (the proof of this relies on standard concentration bounds). This effectively lifts the cost of the computation from being super-polynomial in w to being only super-polynomial in $\beta 2^k \log w$, which remains polynomial overall when setting β and k to be appropriately small.

2.3 Application to Sublinear Secure Computation

The work of [Cou19] gives a generic secure protocol with sublinear communication for layered circuits. It works in the *corruptible correlated randomness model*: before the protocol, a trusted dealer lets the adversary choose the strings that the corrupted parties will get, samples the correlated random coins of the remaining parties afterwards, and distributes them to the parties. As shown in [BCG+19b], generating this corruptible randomness using a PCG leads to a secure protocol in the standard model. In a bit more detail, the parties use a generic secure protocol to generate the short seeds $(\mathsf{k}_0, \mathsf{k}_1)$ then expand them locally; it might have a high overhead, but it will not be a bottleneck since the seeds are very small. We show that our new PCG can be used for just this purpose.

The general idea is to split a layered circuit of size s into carefully chosen blocks, each containing $O(\log\log s)$ consecutive layers. The precise block decomposition is detailed in [Cou19]. Using our PCG cast as an HSS scheme for $O(\log\log s)$-depth circuits (with the duality described in section 2.1) allows the parties to evaluate the circuit in a block-by-clock fashion: for each block the parties start with additive shares of

– the inputs of the circuit;
– the values of the first layer of the block;

and, using HSS, compute additive shares of

– the outputs of the circuit which are in the block;
– the values of the last layer, which are also the values of the first layer of the next block.

Let us note that since the circuit and its blocks are publicly known to both parties, so the fact our HSS scheme is circuit-dependent is not an issue here. This block-by-block approach allows the parties to 'skip' a fraction $O(\log\log(s))$ of the gates when computing the circuit, by communicating at each block rather than at each gate. Unfortunately, combining all these blocks together involves pesky technicalities which prohibit a very modular approach and require us to consider the protocol in its entirety. Indeed, the inputs can appear arbitrarily many times–up to $O(s)$ even–across many blocks, so the randomness used to mask them has to be reused, and we cannot deal with each block using an independent instance of HSS. However, dealing with this problem does not require any additional insight, only more cumbersome notations.

In the above outline, we assumed that we had access to a sufficiently low-communication MPC protocol to distribute the generation of the seeds to our new PCG. To obtain our claimed result, it remains to show that this building block can be instantiated under the quasi-polynomial hardness of LPN. In fact, this MPC protocol needs not have linear communication in the seed size; it turns out that by tuning the parameters appropriately, any fixed polynomial in the seed size suffices to guarantee the existence of a "soft spot" for the parameters of our PCG such that we simultaneously get sublinear total communication $O(s/\log\log s)$ and polynomial computation. Distributing the generation procedure of our PCG essentially boils down to generating (many) seeds for a multi-point function secret sharing scheme, which itself boils down mainly to securely generating seeds for a standard length-doubling pseudorandom generator (PRG), and securely executing about $\log(\mathsf{domsize})$ expansions of these short seeds, where $\mathsf{domsize}$ denotes the domain size of the MPFSS. Using a standard LPN-based PRG and GMW-style secure computation, instantiated with an LPN-based oblivious transfer protocol, suffices to securely generate the MPFSS seeds we need.

3 Preliminaries

3.1 Function Secret Sharing

Informally, an FSS scheme for a class of functions $\mathscr{C}$ is a pair of algorithms $\mathsf{FSS} = (\mathsf{FSS.Gen}, \mathsf{FSS.Eval})$ such that:

– $\mathsf{FSS.Gen}$ given a function $f \in \mathscr{C}$ outputs a pair of keys (K_0, K_1);
– $\mathsf{FSS.Eval}$, given K_b and input x, outputs y_b such that y_0 and y_1 form additive shares of $f(x)$.

The security requirement is that each key K_b computationally hide f, except for revealing the input and output domains of f. Formally:

Definition 3 (Function Secret Sharing; adapted from [BGI16b]). *A 2-party* function secret sharing *(FSS) scheme for a class of functions $\mathscr{C} = \{f : I \to \mathbb{G}\}$ with input domain I and output domain an abelian group $(\mathbb{G}, +)$, is a pair of PPT algorithms* $\mathsf{FSS} = (\mathsf{FSS.Gen}, \mathsf{FSS.Eval})$ *with the following syntax:*

- $\mathsf{FSS.Gen}(1^\lambda, f)$, *given security parameter* λ *and description of a function* $f \in \mathscr{C}$, *outputs a pair of keys* (K_0, K_1)*;*
- $\mathsf{FSS.Eval}(b, K_b, x)$, *given party index* $b \in \{0,1\}$, *key* K_b, *and input* $x \in I$, *outputs a group element* $y_b \in \mathbb{G}$.

Given an allowable leakage function $\mathsf{Leak} : \{0,1\}^* \to \{0,1\}^*$, *the scheme* FSS *should satisfy the following requirements:*

- **Correctness:** *For any* $f : I \to \mathbb{G}$ *in* $\mathscr{C}$ *and* $x \in I$, *we have* $\Pr[(K_0, K_1) \xleftarrow{\$} \mathsf{FSS.Gen}(1^\lambda, f) : \sum_{b\in\{0,1\}} \mathsf{FSS.Eval}(b, K_b, x) = f(x)] = 1$.
- **Security:** *For any* $b \in \{0,1\}$, *there exists a PPT simulator* Sim *such that for any polynomial-size function sequence* $f_\lambda \in \mathscr{C}$, *the distributions* $\{(K_0, K_1) \xleftarrow{\$} \mathsf{FSS.Gen}(1^\lambda, f_\lambda) : K_b\}$ *and* $\{K_b \xleftarrow{\$} \mathsf{Sim}(1^\lambda, \mathsf{Leak}(f_\lambda))\}$ *are computationally indistinguishable.*

Our application of FSS requires applying the evaluation algorithm on *all inputs.* Following [BGI16b, BCGI18, BCG+19b, BCG+19a], given an FSS scheme (FSS.Gen, FSS.Eval), we denote by FSS.FullEval an algorithm which, on input a bit b, and an evaluation key K_b (which defines the input domain I), outputs a list of $|I|$ elements of $\mathbb{G}$ corresponding to the evaluation of $\mathsf{FSS.Eval}(b, K_b, \cdot)$ on every input $x \in I$ (in some predetermined order). Below, we recall some results from [BGI16b] on FSS schemes for useful classes of functions.

Distributed Point Functions A distributed point function (DPF) [GI14] is an FSS scheme for the class of point functions $f_{\alpha,\beta} : \{0,1\}^\ell \to \mathbb{G}$ which satisfies $f_{\alpha,\beta}(\alpha) = \beta$, and $f_{\alpha,\beta}(x) = 0$ for any $x \neq \alpha$. A sequence of works [GI14, BGI15, BGI16b] has led to highly efficient constructions of DPF schemes from any pseudorandom generator (PRG).

Theorem 4 (PRG-based DPF [BGI16b]). *Given a PRG* $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda+2}$, *there exists a DPF for point functions* $f_{\alpha,\beta} : \{0,1\}^\ell \to \mathbb{G}$ *with key size* $\ell \cdot (\lambda + 2) + \lambda + \lceil \log_2 |\mathbb{G}| \rceil$ *bits. For* $m = \lceil \frac{\log |\mathbb{G}|}{\lambda+2} \rceil$, *the key generation algorithm* Gen *invokes* G *at most* $2(\ell + m)$ *times, the evaluation algorithm* Eval *invokes* G *at most* $\ell + m$ *times, and the full evaluation algorithm* FullEval *invokes* G *at most* $2^\ell(1 + m)$ *times.*

FSS for Multi-Point Functions Similarly to [BCGI18, BCG+19b, BCG+19a], we use FSS for *multi-point functions.* A k-point function evaluates to 0 everywhere, except on k specified points. When specifying multi-point functions we often view the domain of the function as $[n]$ for $n = 2^\ell$ instead of $\{0,1\}^\ell$.

Definition 5 (Multi-Point Function [BCGI18]). *An* (n, t)*-multi-point function over an abelian group* $(\mathbb{G}, +)$ *is a function* $f_{S,\boldsymbol{y}} : [n] \to \mathbb{G}$, *where* $S = (s_1, \cdots, s_t)$ *is an ordered subset of* $[n]$ *of size* t *and* $\boldsymbol{y} = (y_1, \cdots, y_t) \in \mathbb{G}^t$, *defined by* $f_{S,\boldsymbol{y}}(s_i) = y_i$ *for any* $i \in [t]$, *and* $f_{S,y}(x) = 0$ *for any* $x \in [n] \setminus S$.

We assume that the description of S includes the input domain $[n]$ so that $f_{S,\boldsymbol{y}}$ is fully specified. A *Multi-Point Function Secret Sharing* (MPFSS) is an FSS scheme for the class of multi-point functions, where a point function $f_{S,\boldsymbol{y}}$ is represented in a natural way. We assume that an MPFSS scheme leaks not only the input and output domains but also the number of points t that the multi-point function specifies. An MPFSS can be easily obtained by adding t instances of a DPF.

3.2 Learning Parity with Noise

Our constructions rely on the Learning Parity with Noise assumption [BFKL93] (LPN) over a field $\mathbb{F}$ (the most standard variant of LPN typically assumes $\mathbb{F} = \mathbb{F}_2$, but other fields can be considered). Unlike the LWE assumption, in LPN over $\mathbb{F}$ the noise is assumed to have a small Hamming weight. Concretely, the noise is a random field element in a small fraction of the coordinates and 0 elsewhere. Given a field $\mathbb{F}$, $\mathsf{Ber}_r(\mathbb{F})$ denote the distribution which outputs a uniformly random element of $\mathbb{F} \setminus \{0\}$ with probability r, and 0 with probability $1 - r$.

Definition 6 (LPN). *For dimension $k = k(\lambda)$, number of samples (or block length) $q = q(\lambda)$, noise rate $r = r(\lambda)$, and field $\mathbb{F} = \mathbb{F}(\lambda)$, the $\mathbb{F}$-$\mathsf{LPN}(k, q, r)$ assumption states that*

$$\begin{aligned} &\{(A, \boldsymbol{b}) \mid A \xleftarrow{\$} \mathbb{F}^{q\times k}, \boldsymbol{e} \xleftarrow{\$} \mathsf{Ber}_r(\mathbb{F})^q, \boldsymbol{s} \xleftarrow{\$} \mathbb{F}^k, \boldsymbol{b} \leftarrow A \cdot \boldsymbol{s} + \boldsymbol{e}\} \\ \overset{c}{\approx}&\{(A, \boldsymbol{b}) \mid A \xleftarrow{\$} \mathbb{F}^{q\times k}, \boldsymbol{b} \xleftarrow{\$} \mathbb{F}^q\} \end{aligned}$$

Here and in the following, all parameters are functions of the security parameter λ and computational indistinguishability is defined with respect to λ. Note that the search LPN problem, of finding the vector can be reduced to the decisional LPN assumption [BFKL93, AIK09]. In this paper, our protocols will mostly rely on a variant of LPN, called *exact LPN* (xLPN) [JKPT12]. In this variant, the noise vector $\boldsymbol{e}$ is not sampled from $\mathsf{Ber}_r(\mathbb{F})^q$, but it is sampled uniformly from the set $\mathsf{HW}_{rq}(\mathbb{F}^q)$ of length-q vectors over $\mathbb{F}$ with *exactly* rq nonzero coordinates (in contrast, a sample from $\mathsf{Ber}_r(\mathbb{F})^q$ has an *expected* number $r \cdot q$ of nonzero coordinates). While standard LPN is usually preferred since the Bernouilli distribution is convenient to analyze, xLPN is often preferred in concrete implementations, since it offers a potentially higher level of security for similar parameters (by avoiding weak instances with a low amount of noise). Furthermore, as outlined in [JKPT12], xLPN and LPN are equivalent: xLPN reduces to its search version using the sample-preserving reduction of [AIK07], and search-xLPN is easily seen to be polynomially equivalent to search-LPN.

Dual LPN. In our protocols, it will also prove convenient to work with the (equivalent) alternative *dual* formulation of LPN.

Definition 7 (Dual LPN). *For dimension $k = k(\lambda)$, number of samples (or block length) $q = q(\lambda)$, noise rate $r = r(\lambda)$, and field $\mathbb{F} = \mathbb{F}(\lambda)$, the dual-$\mathbb{F}$-$\mathsf{LPN}(k, q, r)$ assumption states that*

$$\begin{aligned} &\{(H, \boldsymbol{b}) \mid H \xleftarrow{\$} \mathbb{F}^{q-k\times q}, \boldsymbol{e} \xleftarrow{\$} \mathsf{Ber}_r(\mathbb{F})^q, \boldsymbol{b} \leftarrow H \cdot \boldsymbol{e}\} \\ \overset{c}{\approx}&\{(H, \boldsymbol{b}) \mid H \xleftarrow{\$} \mathbb{F}^{q-k\times q}, \boldsymbol{b} \xleftarrow{\$} \mathbb{F}^q\} \end{aligned}$$

Solving the dual LPN assumption is easily seen to be at least as hard as solving LPN: given a sample $(A, \boldsymbol{b})$, define $H \in \mathbb{F}^{q-k\times q}$ to be the parity-check matrix of A (hence $H \cdot A = 0$), and feed $(H, H \cdot \boldsymbol{b})$ to the dual LPN solver. Note that the parity check matrix of a random matrix is distributed as a random matrix. Furthermore, when $\boldsymbol{b} = A \cdot \boldsymbol{s} + \boldsymbol{e}$, we have $H \cdot \boldsymbol{b} = H \cdot (A \cdot \boldsymbol{s} + \boldsymbol{e}) = H \cdot \boldsymbol{e}$. For discussions regarding existing attacks on LPN and their efficiency, we refer the reader to [BCGI18, BCG+19b].

3.3 Pseudorandom Correlation Generators

Pseudorandom correlation generators (PCG) have been introduced in [BCG+19b]. Informally, a pseudorandom correlation generator allows to generate pairs of short keys (or seeds) $(\mathsf{k}_0, \mathsf{k}_1)$ such that each key k_σ can be expanded to a long string $R_\sigma = \mathsf{Expand}(\sigma, \mathsf{k}_\sigma)$, with the following guarantees: given the key $\mathsf{k}_{1-\sigma}$, the string R_σ is indistinguishable from a random string sampled conditioned on satisfying the target correlation with the string $R_{1-\sigma} = \mathsf{Expand}(1-\sigma, \mathsf{k}_{1-\sigma})$. We provide below the formal definition of pseudorandom correlation generators, after defining the notion of *reverse-sampleable correlation generator.*

Definition 8 (Correlation Generator). *A PPT algorithm $\mathscr{C}$ is called a* correlation generator*, if $\mathscr{C}$ on input 1^λ outputs a pair of elements in $\{0, 1\}^n \times \{0, 1\}^n$ for $n \in \mathsf{poly}(\lambda)$.*

In order to define security, we require the notion of a reverse-sampleable correlation generator introduced in the following.

Definition 9 (Reverse-sampleable Correlation Generator). *Let $\mathscr{C}$ be a correlation generator. We say $\mathscr{C}$ is* reverse sampleable *if there exists a PPT algorithm $\mathsf{RSample}$ such that for $\sigma \in \{0, 1\}$ the correlation obtained via:*

$$\{(R'_0, R'_1) \mid (R_0, R_1) \xleftarrow{\$} \mathscr{C}(1^\lambda), R'_\sigma := R_\sigma, R'_{1-\sigma} \xleftarrow{\$} \mathsf{RSample}(\sigma, R_\sigma)\}$$

is computationally indistinguishable from $\mathscr{C}(1^\lambda)$.

Definition 10 (Pseudorandom Correlation Generator (PCG)).
Let $\mathscr{C}$ be a reverse-sampleable correlation generator. A pseudorandom correlation generator (PCG) for *$\mathscr{C}$ is a pair of algorithms* (PCG.Gen, PCG.Expand) *with the following syntax:*

– $\mathsf{PCG.Gen}(1^\lambda)$ *is a PPT algorithm that given a security parameter λ, outputs a pair of seeds* $(\mathsf{k}_0, \mathsf{k}_1)$*;*
– $\mathsf{PCG.Expand}(\sigma, \mathsf{k}_\sigma)$ *is a polynomial-time algorithm that given party index $\sigma \in \{0,1\}$ and a seed k_σ, outputs a bit string $R_\sigma \in \{0,1\}^n$.*

The algorithms (PCG.Gen, PCG.Expand) *should satisfy the following:*

– **Correctness.** *The correlation obtained via:*

$$\{(R_0, R_1) \mid (\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{PCG.Gen}(1^\lambda), R_\sigma \leftarrow \mathsf{PCG.Expand}(\sigma, \mathsf{k}_\sigma) \textit{ for } \sigma \in \{0,1\}\}$$

is computationally indistinguishable from $\mathscr{C}(1^\lambda)$.
– **Security.** *For any $\sigma \in \{0,1\}$, the following two distributions are computationally indistinguishable:*

$$\{(\mathsf{k}_{1-\sigma}, R_\sigma) \mid (\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{PCG.Gen}(1^\lambda), R_\sigma \leftarrow \mathsf{PCG.Expand}(\sigma, \mathsf{k}_\sigma)\} \textit{ and}$$
$$\{(\mathsf{k}_{1-\sigma}, R_\sigma) \mid (\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{PCG.Gen}(1^\lambda), R_{1-\sigma} \leftarrow \mathsf{PCG.Expand}(\sigma, \mathsf{k}_{1-\sigma}),$$
$$R_\sigma \xleftarrow{\$} \mathsf{RSample}(\sigma, R_{1-\sigma})\}$$

where RSample *is the reverse sampling algorithm for correlation $\mathscr{C}$.*

Note that the above definition is trivial to achieve in general: We can let PCG.Gen on input 1^λ return $(R_0, R_1) \leftarrow \mathscr{C}(1^\lambda)$, and simply define Expand to be the identity. Typically, we will be interested in non-trivial constructions of PCGs, in which the seed size is significantly shorter than the output size. A pseudorandom generator with image in $\{0,1\}^n$ is a simple example for an expanding PCG for the equality correlation $\{(R, R) \mid R \in \{0,1\}^n\}$.

4 Secure Computation from Super-Constant-Degree Low-Locality Polynomial Correlated Randomness

4.1 Block Decomposition of Layered Circuits

Given an arithmetic circuit C and an input vector $\boldsymbol{x}$, we call *value of the gate g on input $\boldsymbol{x}$* the value carried by the output wire of a given gate g of C during the evaluation of $C(\boldsymbol{x})$. The following decomposition of layered circuits is implicit in [Cou19]; for completeness, we give the proof here.

Lemma 11 (Block-Decomposition of Layered Circuits, Implicit in [Cou19]). *Let C be a layered arithmetic circuit over a field $\mathbb{F}$ with n inputs and m outputs, of size s and depth $d = d(n)$. For any integer k, denoting $t = t(k) = \lceil d/k \rceil$, there exists $2t+1$ integers $(s_0 = 0, s_1, \cdots, s_{t-1}, s_t = 0)$, $(m_0, \cdots, m_{t-1})$, and functions $(f_0, \cdots, f_{t-1})$ with $f_i : \mathbb{F}^n \times \mathbb{F}^{s_i} \to \mathbb{F}^{s_{i+1}} \times \mathbb{F}^{m_i}$, such that:*

– *The algorithm A given below satisfies, for any input vector $\boldsymbol{x} \in \mathbb{F}^n$, $A(\boldsymbol{x}) = C(\boldsymbol{x})$ (that is, A computes C);*

```
function A(x)
    $\boldsymbol{x}_0 \leftarrow \boldsymbol{x}$
    for $i = 0$ to $t-1$ do $(\boldsymbol{x}_{i+1}, \boldsymbol{y}_i) \leftarrow f_i(\boldsymbol{x}_i)$
    $\boldsymbol{y} \leftarrow \boldsymbol{y}_0 || \cdots || \boldsymbol{y}_{t-1}$
    return $\boldsymbol{y}$
```

– *For any $i \in [\![0, t-1]\!]$, $j \leq s_{i+1} + m_i$, the j-th output[5] of $f_i : \mathbb{F}^n \times \mathbb{F}^{s_i} \mapsto \mathbb{F}^{s_{i+1}} \times \mathbb{F}^{m_i}$ can be computed by a multivariate polynomial $P_{i,j}$ over $\mathbb{F}^{2^k}$ of degree $\deg P_{i,j} \leq 2^k$;*
– *$\sum_{i=0}^{t-1} s_i \leq s/k$ and $\sum_{i=0}^{t-1} m_i = m$.*

The decomposition of a layered circuits into chunks computable by low-degree functions is illustrated on Figure 1.

[5] *i.e.* the j^{th} coordinate of the image by f_i, seen as $f_i \colon \mathbb{F}^n \times \mathbb{F}^{s_i} \to \mathbb{F}^{s_{i+1}+m_i}$.

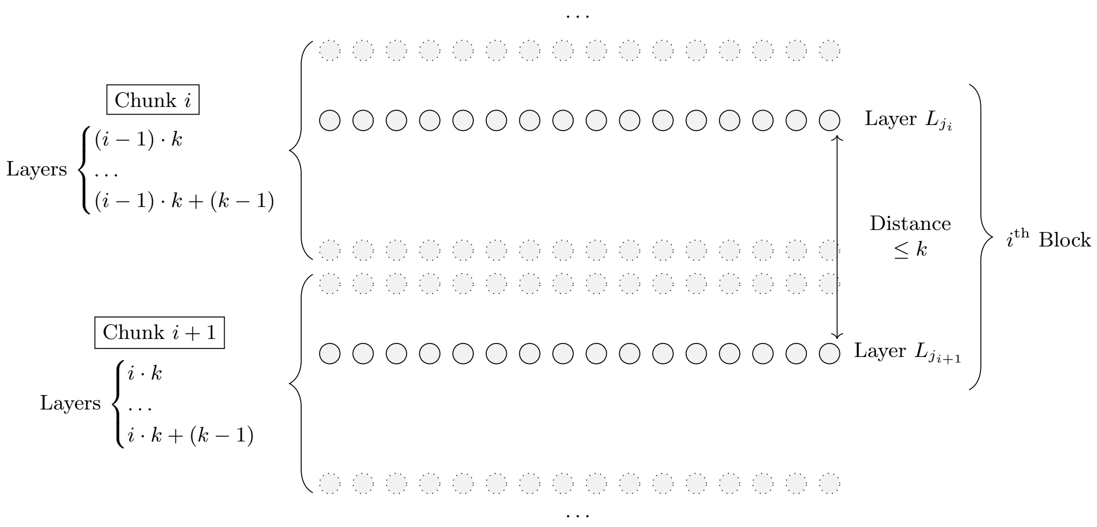

Fig. 1: Block Decomposition of a Circuit.

Proof. Let C be a layered boolean circuit with n inputs and m outputs, of size s and depth d, with layers $(L_1, \cdots, L_d)$. For $i = 1$ to d, we let w_i denote the width of the layer L_i (that is, the number of computation gates it contains; note that $s = \sum_{i=1}^{d} w_i$). Fix an integer k and let $t = \lceil d/k \rceil$.

We start by considering t 'chunks' of layers, each containing k consecutive layers (the last may in fact contain fewer if $k \nmid d$). Observe there must exist a $j \in \{0, \dots, k-1\}$, such that the sum of the widths of the j^{th} layer of each chunk (with the convention that if the last chunk has fewer than j layer, its j^{th} one is empty; $w_i = 0$ if $i > d$) is at most s/k, *i.e.* $\sum_{i=1}^{t} w_{k\cdot(i-1)+j} \leq s/k$. Indeed otherwise $\forall j \in \{0, \dots, k\}, \sum_{i=0}^{t-1} w_{r_0+1+k\cdot(i-1)+j} > s/k$, so $s = \sum_{i=1}^{d} w_i \geq \sum_{i=r_0+1}^{d-r_1} w_i = \sum_{j=0}^{k-1}\sum_{i=1}^{t} w_{r_0+1+k\cdot(i-1)+j} > k \cdot s/k = s$, which is a contradiction. With j being fixed we now define $j_i \leftarrow k\cdot(i-1)+j$ for $i \in [t-1]$ and $j_t \leftarrow \min(d,\ k\cdot(t-1)+j)$.

Now, for each $0 \leq i \leq t$, we let B_i the block containing the consecutive layers $(L_{j_i}, \cdots, L_{j_{i+1}})$. Note that the depth of each block is at most k. Let m_i denote the number of output nodes contained in B_i (note that $\sum_{i=0}^{t-1} m_i = m$). For each $1 \leq i \leq t$, set $s_i \leftarrow w_{j_i}$, and $s_0, s_d \leftarrow 0$. This decomposition into blocks is illustrated in fig. 1.

The intuition behind the decomposition of C is the following: each function f_i will take as input the n inputs $\boldsymbol{x}$ to C, together with the s_i values of the gates in the first layer of B_i. It evaluates the layers of B_i, starting from the s_i values of the first layer (using the input $\boldsymbol{x}$ when the layer contains an input node), and outputs the s_{i+1} values on the first layer of B_{i+1}, together with the m_i values of the output nodes contained in B_i. Given this decomposition, the correctness of algorithm A is guaranteed by definition: A simply corresponds to a "block-by-block" evaluation of the circuit C, where each block evaluation outputs the current state $\boldsymbol{x}_{i+1}$ (which must be given as input to the next block in addition to the input vector $\boldsymbol{x}$) together with the outputs of C contained in this block. Since each block has depth at most k, each output of f_i can be computed by multivariate polynomials with at most 2^k inputs, and of degree at most 2^k. □

4.2 Securely Computing C in the Correlated Randomness Model

We represent in fig. 2 the ideal functionality for securely evaluating the layered arithmetic circuit C.

We represent on fig. 3 an ideal functionality for distributing (function-dependent) correlated randomness between the parties.

Theorem 12. *Let $k \leq \log\log s - \log\log\log s$. There exists a protocol Π_C which (perfectly) securely implements the N-party functionality $\mathscr{F}_C$ in the $\mathscr{F}_{\mathsf{corr}}$-hybrid model, against a static, passive, non-aborting adversary corrupting at most $N-1$ out of N parties, with communication complexity upper bounded by $O(N\cdot(n+\frac{s}{k}+m)\cdot\log|\mathbb{F}|)$ and polynomial computation.*

The protocol follows closely the construction of [Cou19], with some tedious technical adaptations which are necessary to rely on the specific type of correlated randomness which we will manage to

Ideal Functionality $\mathscr{F}_C$

- **Parameters.** The functionality is parametrised with an arithmetic circuit C with n inputs over a finite field $\mathbb{F}$.
- **Parties.** An adversary $\mathscr{A}$ and N parties $P_1, \cdots, P_N$. Each party P_ℓ has $p_\ell \in [0, n]$ inputs over $\mathbb{F}$, with $\sum_{\ell \leq N} p_\ell = n$.

The functionality aborts if it receives any incorrectly formatted message.

1. On input a message $(\mathsf{input}, \boldsymbol{x}_\ell)$ from each party P_ℓ where $\boldsymbol{x}_\ell \in \mathbb{F}^{p_\ell}$, set
$$\boldsymbol{x} \leftarrow \boldsymbol{x}_1 || \cdots || \boldsymbol{x}_N \in \mathbb{F}^n.$$
2. Compute $\boldsymbol{y} \leftarrow C(\boldsymbol{x})$. Output $\boldsymbol{y}$ to all parties, and terminate.

Fig. 2: Ideal functionality $\mathscr{F}_C$ for securely evaluating an arithmetic circuit C among N parties.

Ideal Functionality $\mathscr{F}_{\mathsf{corr}}$

- **Parameters.** For every $i = 0, \ldots, \lceil d/k \rceil - 1$, functionality is parameterised with subsets $(U^{\mathsf{in}}_{i,j}, U_{i,j})_{1 \leq j \leq \lceil s_{i+1}/\beta \rceil}$ and $(V^{\mathsf{in}}_{i,j}, V_{i,j})_{1 \leq j \leq \lceil m_i/\beta \rceil}$.
- **Parties.** An adversary $\mathscr{A}$ and N parties $P_1, \cdots, P_N$.

The functionality aborts if it receives any incorrectly formatted message.

1. On input a message $(\mathsf{corrupt}, D)$ with $D \subsetneq [N]$ from $\mathscr{A}$, set $H \leftarrow [N] \setminus D$ and store (H, D).
2. On input a message input with from each party P_ℓ, send ready to $\mathscr{A}$.
3. *Setup input masks:* On input a message $(\mathsf{setinputshare}, (\boldsymbol{r}_{\mathsf{in},\ell})_{\ell \in D})$ from $\mathscr{A}$ with $\forall \ell \in D, \boldsymbol{r}_{\mathsf{in},\ell} \in \mathbb{F}^n$, sample $(\boldsymbol{r}_{\mathsf{in},\ell})_{\ell \in H} \overset{\$}{\leftarrow} (\mathbb{F}^n)^{|H|}$, and set $\boldsymbol{r}_{\mathsf{in}} \leftarrow \sum_{\ell \in [N]} \boldsymbol{r}_{\mathsf{in},\ell}$.
4. For $i = 1$ to $\lceil d/k \rceil - 1$:
   (a) *Setup masks for the computation gates of the first layer of the i^{th} chunk:* On input a message $(\mathsf{setblockshare}, i, (\boldsymbol{r}_{i,\ell})_{\ell \in D})$ from $\mathscr{A}$ with $\forall \ell \in D, \boldsymbol{r}_{i,\ell} \in \mathbb{F}^{s_i}$, sample $(\boldsymbol{r}_{i,\ell})_{\ell \in H} \overset{\$}{\leftarrow} (\mathbb{F}^{s_i})^{|H|}$, and set $\boldsymbol{r}_{\mathsf{in}} \leftarrow \sum_{\ell \in [N]} \boldsymbol{r}_{\mathsf{in},\ell}$.
   (b) *Setup evaluation of the computation gates on the final layer of the i^{th} chunk:*
      - For $j = 1$ to $\lceil s^{i+1}/\beta \rceil$, set:
$$\boldsymbol{\pi}^{(i,j)} \leftarrow \left(1 \;\Big|\Big|\; \boldsymbol{r}_{\mathtt{in}}[U^{\mathtt{in}}_{i,j}] \;\Big|\Big|\; \boldsymbol{r}_i[U_{i,j}]\right)^{\otimes 2^k}.$$
      - Wait for a message $(\mathsf{setshare}, (i,j), (\boldsymbol{\pi}^{(i,j)}_\ell)_{\ell \in D})$ from $\mathscr{A}$ with $\boldsymbol{\pi}^{(i,j)}_\ell \in \mathbb{F}^\delta$;
      - Compute uniformly random shares $(\boldsymbol{\pi}^{(i,j)}_\ell)_{\ell \in |H|}$ of $\boldsymbol{\pi}^{(i,j)} - \sum_{\ell \in D} \boldsymbol{\pi}^{(i,j)}_\ell$.
   (c) *Setup evaluation of the output gates in the i^{th} chunk:*
      - For $j = 1$ to $\lceil m_i/\beta \rceil$, set:
$$\boldsymbol{\pi}^{(i,j)} \leftarrow \left(1 \;\Big|\Big|\; \boldsymbol{r}_{\mathtt{in}}[V^{\mathtt{in}}_{i,j}] \;\Big|\Big|\; \boldsymbol{r}_i[V_{i,j}]\right)^{\otimes 2^k}.$$
      - Wait for a message $(\mathsf{setoutputshare}, (i,j), (\boldsymbol{\pi}^{(i,j)}_\ell)_{\ell \in D})$ from $\mathscr{A}$ with $\boldsymbol{\pi}^{(i,j)}_\ell \in \mathbb{F}^\delta$;
      - Compute uniformly random shares $(\boldsymbol{\pi}^{(i,j)}_\ell)_{\ell \in |H|}$ of $\boldsymbol{\pi}^{(i,j)} - \sum_{\ell \in D} \boldsymbol{\pi}^{(i,j)}_\ell$.
5. Output $(\boldsymbol{r}_{\mathsf{in},\ell}, (\boldsymbol{r}_{i,\ell}, (\boldsymbol{\pi}^{(i,j)}_\ell)_{1 \leq j \leq \lceil s_{i+1}/\beta \rceil}, (\boldsymbol{\pi}^{(i,j)}_{\mathsf{out},\ell})_{1 \leq j \leq \lceil m_i/\beta \rceil})_{0 \leq i < \lceil d/k \rceil})$ to each party P_ℓ.

Fig. 3: Ideal corruptible functionality $\mathscr{F}_{\mathsf{corr}}$ to deal out correlated randomness to the parties.

securely generate with low communication overhead. The rest of this section is dedicated to making Π_C explicit and to analysing its security.

In the sequel, we fix a layered arithmetic C with block decomposition:

$$(s_0, s_1, \cdots, s_{t-1}), (m_0, \cdots, m_{t-1}), \text{ and } (f_0, \cdots, f_{t-1}).$$

We now proceed with the description of the protocol Π_C, which securely implements $\mathscr{F}_C$ in the $\mathscr{F}_{\mathsf{corr}}$-hybrid model, with security against a static adversary passively corrupting at most $N-1$ parties. Fix parameters $\beta, k \in \mathbb{N}^*$. We let $\boldsymbol{x}_\ell$ denote the input vector of party P_ℓ over $\mathbb{F}$. We slightly abuse this notation and view each vector $\boldsymbol{x}_\ell$ as a length-n vector with zeroes at the positions were P_ℓ does not hold an input, so that the vectors $\boldsymbol{x}_\ell$ form additive shares of the input vector $\boldsymbol{x} = \sum_{\ell=1}^{N} \boldsymbol{x}_\ell$.

For each block B_i, we denote by $\boldsymbol{u}^i$ the values on the nodes of the first layer of B_i, and by $\boldsymbol{v}^i$ the values on all output nodes in B_i. Observe that by definition of f_i, we have $f_i(\boldsymbol{x}, \boldsymbol{u}^i) = (\boldsymbol{u}^{i+1}, \boldsymbol{v}^i)$. We further partition the outputs of f_i in subvectors $(\boldsymbol{u}_j^{i+1})_{1\leq j\leq \lceil s_{i+1}/\beta \rceil}$ and $(\boldsymbol{v}_j^{i})_{1\leq j\leq \lceil m_{i}/\beta \rceil}$, such that each subvector has length at most β. Recall that each output of f_i depends on at most 2^k inputs; therefore, each subvector $\boldsymbol{u}_j^{i+1}$ and $\boldsymbol{v}_j^i$ depends on at most $\beta \cdot 2^k$ coordinates of $(\boldsymbol{x}, \boldsymbol{u}^i)$. For each subvector $\boldsymbol{u}_j^{i+1}$ (resp. $\boldsymbol{v}_j^i$), we denote by $U_{i,j}^{\mathsf{in}} \subset [n]$ (resp. $V_{i,j}^{\mathsf{in}} \subset [n]$) the subset of coordinates of $\boldsymbol{x}$ which influence $\boldsymbol{u}_j^{i+1}$ (resp. $\boldsymbol{v}_j^i$), and by $U_{i,j} \subset [s_i]$ (resp. $V_{i,j} \subset [s_i]$) the subset of coordinates of $\boldsymbol{u}^i$ which influence $\boldsymbol{u}_j^{i+1}$ (resp. $\boldsymbol{v}_j^i$). Note that $|U_{i,j}^{\mathsf{in}}| + |U_{i,j}| \leq \beta \cdot 2^k$ and $|V_{i,j}^{\mathsf{in}}| + |V_{i,j}| \leq \beta \cdot 2^k$. This decomposition is illustrated in fig. 4.

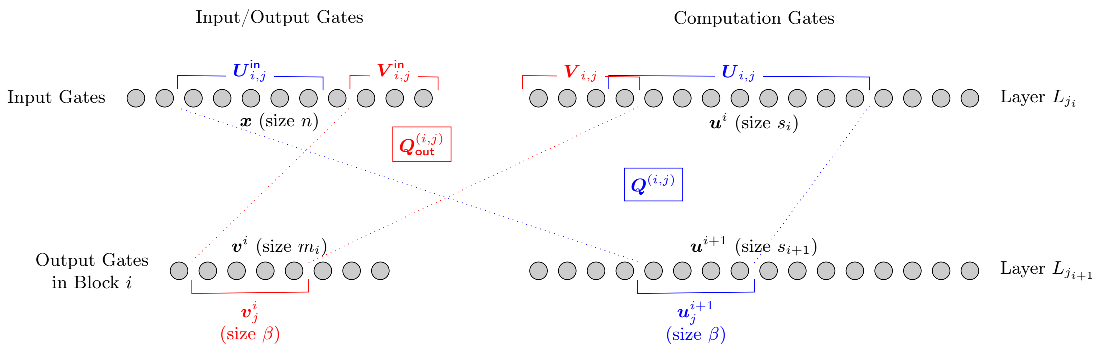
Input/Output Gates
Computation Gates
Input Gates
$U_{i,j}^{\mathsf{in}}$
$V_{i,j}^{\mathsf{in}}$
$\boldsymbol{x}$ (size n)
$Q_{\mathsf{out}}^{(i,j)}$
$V_{i,j}$
$U_{i,j}$
Layer L_{j_i}
$\boldsymbol{u}^i$ (size s_i)
$Q^{(i,j)}$
Output Gates in Block i
$\boldsymbol{v}^i$ (size m_i)
$\boldsymbol{u}^{i+1}$ (size s_{i+1})
Layer $L_{j_{i+1}}$
$\boldsymbol{v}_j^i$ (size β)
$\boldsymbol{u}_j^{i+1}$ (size β)

Fig. 4: Decomposition of the i^{th} Block into low-degree Polynomials.

Initialisation.

– Each party P_ℓ sends input to $\mathscr{F}_{\mathsf{corr}}$ and waits until it receives

$$(\boldsymbol{r}_{\mathsf{in},\ell}, (\boldsymbol{r}_{i,\ell}, (\boldsymbol{\pi}_\ell^{(i,j)})_{1\leq j\leq\lceil s_{i+1}/\beta\rceil}, (\boldsymbol{\pi}_{\mathsf{out},\ell}^{(i,j)})_{1\leq j\leq\lceil m_i/\beta\rceil})_{0\leq i<t}).$$

– Each party P_ℓ broadcasts $\boldsymbol{z}_\ell^{\mathsf{in}} \leftarrow \boldsymbol{x}_\ell + \boldsymbol{r}_{\mathsf{in},\ell}$. All parties compute $\boldsymbol{z}^{\mathsf{in}} \leftarrow \sum_\ell \boldsymbol{z}_\ell^{\mathsf{in}} = \boldsymbol{x} + \boldsymbol{r}_{\mathsf{in}}$.

***i*-th Block Evaluation.** We now resume the description of the protocol, and assume that the protocol maintains the following invariant: at the beginning of the i-th block evaluation, each party P_ℓ holds an additive share $\boldsymbol{u}_\ell^i$ of the values $\boldsymbol{u}^i$ on the first layer of the block. The block decomposition of f_i guarantees that each output of $f_i : \mathbb{F}^n \times \mathbb{F}^{s_i} \mapsto \mathbb{F}^{s_{i+1}} \times \mathbb{F}^{m_i}$ can be computed by a multivariate polynomial of degree at most 2^k. Let $\boldsymbol{Q}^{(i,j)}$ denote the vector of multivariate polynomials of degree at most 2^k such that $\boldsymbol{Q}^{(i,j)}(\boldsymbol{x}[U_{i,j}^{\mathsf{in}}] \mid\mid \boldsymbol{u}^i[U_{i,j}]) = \boldsymbol{u}_j^{i+1} \in \mathbb{F}^{s_{i+1}}$. Similarly, let $\boldsymbol{Q}_{\mathsf{out}}^{(i,j)}$ denote the vector of multivariate polynomials of degree at most 2^k such that $\boldsymbol{Q}_{\mathsf{out}}^{(i,j)}(\boldsymbol{x}[V_{i,j}^{\mathsf{in}}] \mid\mid \boldsymbol{u}^i[V_{i,j}]) = \boldsymbol{v}_j^i \in \mathbb{F}^{m_i}$.

– Each party P_ℓ broadcasts $\boldsymbol{z}_\ell^i \leftarrow \boldsymbol{u}_\ell^i + \boldsymbol{r}_{i,\ell}$. All parties compute $\boldsymbol{z}^i \leftarrow \sum_\ell \boldsymbol{z}_\ell = \boldsymbol{u}^i + \boldsymbol{r}_{i,\ell}$.
– Define the following vectors of multivariate polynomials:

$$\hat{\boldsymbol{Q}}^{(i,j)}(\boldsymbol{X}) \leftarrow \boldsymbol{Q}^{(i,j)}(\boldsymbol{X} - (\boldsymbol{r}_{\mathsf{in}}[U_{i,j}^{\mathsf{in}}] \mid\mid \boldsymbol{r}_i[U_{i,j}]))$$
$$\hat{\boldsymbol{Q}}_{\mathsf{out}}^{(i,j)}(\boldsymbol{X}) \leftarrow \boldsymbol{Q}_{\mathsf{out}}^{(i,j)}(\boldsymbol{X} - (\boldsymbol{r}_{\mathsf{in}}[V_{i,j}^{\mathsf{in}}] \mid\mid \boldsymbol{r}_i[V_{i,j}])).$$

Observe that each coefficient of $\hat{\boldsymbol{Q}}^{(i,j)}(\boldsymbol{X})$ is itself a multivariate polynomial of degree at most 2^k in the coordinates of $(\boldsymbol{r}_{\mathsf{in}}[U^{\mathsf{in}}_{i,j}] \mid\mid \boldsymbol{r}_i[U_{i,j}])$. Therefore, all its coefficients can be computed as linear combinations of the coordinates of $\boldsymbol{\pi}^{(i,j)} = (1_{\mathbb{F}} \mid\mid \boldsymbol{r}_{\mathsf{in}}[U^{\mathsf{in}}_{i,j}] \mid\mid \boldsymbol{r}_i[U_{i,j}])^{\otimes 2^k}$. Similarly, all coefficients of $\hat{\boldsymbol{Q}}^{(i,j)}_{\mathsf{out}}(\boldsymbol{X})$ can be computed as linear combinations of the coordinates of $\pi^{(i,j)}_{\mathsf{out}} = (1_{\mathbb{F}} \mid\mid \boldsymbol{r}_{\mathsf{in}}[V^{\mathsf{in}}_{i,j}] \mid\mid \boldsymbol{r}_i[V_{i,j}])^{\otimes 2^k}$.

– Each party P_ℓ computes shares $(\hat{\boldsymbol{Q}}^{(i,j)}_\ell(\boldsymbol{X}), \hat{\boldsymbol{Q}}^{(i,j)}_{\mathsf{out},\ell}(\boldsymbol{X}))$ of $(\hat{\boldsymbol{Q}}^{(i,j)}(\boldsymbol{X}), \hat{\boldsymbol{Q}}^{(i,j)}_{\mathsf{out}}(\boldsymbol{X}))$ from his shares $(\boldsymbol{\pi}^{(i,j)}_\ell, \boldsymbol{\pi}^{(i,j)}_{\mathsf{out},\ell})$ of $(\boldsymbol{\pi}^{(i,j)}, \boldsymbol{\pi}^{(i,j)}_{\mathsf{out}})$, and sets:

$$\boldsymbol{u}^{i+1}_\ell \leftarrow \left(\hat{\boldsymbol{Q}}^{(i,j)}_\ell(\boldsymbol{z}^{\mathsf{in}}[U^{\mathsf{in}}_{i,j}], \boldsymbol{z}^i[U_{i,j}])\right)_{1\le j\le \lceil s_{i+1}/\beta\rceil}$$
$$\boldsymbol{v}^{i}_\ell \leftarrow \left(\hat{\boldsymbol{Q}}^{(i,j)}_{\mathsf{out},\ell}(\boldsymbol{z}^{\mathsf{in}}[V^{\mathsf{in}}_{i,j}], \boldsymbol{z}^i[V_{i,j}])\right)_{1\le j\le \lceil m_i/\beta\rceil}.$$

Output Reconstruction. Each party P_ℓ sets and broadcasts $\boldsymbol{v}_\ell$, where:

$$\boldsymbol{v}_\ell \leftarrow (\boldsymbol{v}^{(0,j)}_\ell)_{j\le q_0} || (\boldsymbol{v}^{(1,j)}_\ell)_{1\le j\le\lceil m_1/\beta\rceil} || \cdots || (\boldsymbol{v}^{(\lceil d/k\rceil-1,j)}_\ell)_{1\le j\le \lceil m_{\lceil d/k\rceil-1}/\beta\rceil}.$$

All parties reconstruct and output $\boldsymbol{v} \leftarrow \sum_\ell \boldsymbol{v}_\ell$.

We now prove Theorem 12 by proving the above protocol satisfies the necessary requirements.

Proof. The efficiency of the protocol follows by inspection: the total length of all messages broadcast in Π_C is

$$N \cdot \left(n + \sum_{i=0}^{t-1} s_i + \sum_{i=0}^{t-1} m_i\right) \le N \cdot \left(n + \frac{s}{k} + m\right).$$

We now analyse the security of Π_C. We describe in fig. 5 a simulator Sim which perfectly simulates an execution of Π_C.

Simulator Sim

Let $D \subsetneq [N]$ be the subset of statically corrupted parties, and $H = [N] \setminus D$ be the (nonempty) subset of honest parties.

– **Initialisation.**
  - Sim honestly simulates the functionality $\mathscr{F}_{\mathsf{corr}}$ and stores the following values:
$$(\boldsymbol{r}_{\mathsf{in},\ell}, (\boldsymbol{r}_{i,\ell}, (\boldsymbol{\pi}^{(i,j)}_\ell)_{1\le j\le\lceil s_{i+1}/\beta\rceil}, (\boldsymbol{\pi}^{(i,j)}_{\mathsf{out},\ell})_{1\le j\le\lceil m_i/\beta\rceil})_{0\le i<\lceil d/k\rceil})_{\ell\in[N]}.$$
  - Sim broadcasts a random vector $\boldsymbol{z}^{\mathsf{in}}_\ell$ on behalf of all honest parties. Let $\boldsymbol{z}^{\mathsf{in}} \leftarrow \sum_{\ell\le N} \boldsymbol{z}^{\mathsf{in}}_\ell$. For each $\ell \in D$, Sim extracts $\boldsymbol{x}_\ell \leftarrow \boldsymbol{z}^{\mathsf{in}}_\ell - \boldsymbol{r}_{\mathsf{in},\ell}$.
  - Sim sends $(\mathsf{input}, \boldsymbol{x}_\ell)$ to $\mathscr{F}_C$ on behalf of all corrupted parties, and receives an output $\boldsymbol{v}$.

– ***i*-th Block Evaluation.**
  Sim broadcasts uniformly random $\boldsymbol{z}^i_\ell \overset{\$}{\leftarrow} \mathbb{F}^{s_i}$ on behalf of all honest parties, waits to receive $(\boldsymbol{z}^i_\ell)_{\ell\in D}$, and sets $\boldsymbol{z}^i \leftarrow \sum_{\ell\in[N]} \boldsymbol{z}^i_\ell$.

– **Output Phase.**
  For each $\ell \in D$, Sim computes $\boldsymbol{v}^i_\ell \leftarrow \hat{\boldsymbol{Q}}^{(i,j)}_{\mathsf{out},\ell}(\boldsymbol{z}^{\mathsf{in}}[V^{\mathsf{in}}_{i,j}], \boldsymbol{z}^i[V_{i,j}])$ and sets $\boldsymbol{v}_\ell \leftarrow \boldsymbol{v}^0_\ell || \cdots || \boldsymbol{v}^{t-1}_\ell$. Eventually, Sim broadcasts uniformly random shares of $\boldsymbol{v} - \sum_{\ell\in D} \boldsymbol{v}_\ell$ on behalf of the honest parties.

Fig. 5: Simulator Sim for the sublinear protocol in the $\mathscr{F}_{\mathsf{corr}}$-hybrid model.

It remains to show that Sim perfectly simulates a run of the real protocol Π_C in the $\mathscr{F}_{\mathsf{corr}}$-hybrid model. But this follows almost immediately by inspection, as the view of all corrupted parties in Π_C contains exactly:

– the tuple $(\boldsymbol{r}_{\mathsf{in},\ell}, (\boldsymbol{r}_{i,\ell}, (\boldsymbol{\pi}^{(i,j)}_\ell)_{1\le j\le\lceil s_{i+1}/\beta\rceil}, (\boldsymbol{\pi}^{(i,j)}_{\mathsf{out},\ell})_{1\le j\le\lceil m_i/\beta\rceil})_{0\le i<t})_{\ell\in D}$, which is chosen by the adversary;

– the vectors $(\boldsymbol{z}_\ell^{\mathsf{in}} = \boldsymbol{x}_\ell + \boldsymbol{r}_{\mathsf{in},\ell})_{\ell \in H}$, which are perfectly random since each $\boldsymbol{r}_{\mathsf{in},\ell}$ is perfectly random, by definition of $\mathscr{F}_{\mathsf{corr}}$;
– the vectors $(\boldsymbol{z}_\ell^i = \boldsymbol{u}_\ell^i + \boldsymbol{r}_\ell^i)_{\ell \in H}$, which are perfectly random since each $\boldsymbol{r}_\ell^i$ is perfectly random, by definition of $\mathscr{F}_{\mathsf{corr}}$;
– the vectors $(\boldsymbol{v}_\ell)_{\ell \in H} = (\boldsymbol{v}_\ell^0 || \cdots || \boldsymbol{v}_\ell^{t-1})_{\ell \in H}$. By construction, and by definition of the $\hat{\boldsymbol{Q}}_{v,\ell}^{(i,j)}(\boldsymbol{X})$, the $\boldsymbol{v}_\ell$ satisfy:

$$\begin{aligned}\boldsymbol{v}_\ell^{(i,j)} &= \hat{\boldsymbol{Q}}_{\mathsf{out},\ell}^{(i,j)}(\boldsymbol{w}[V_{i,j}^{\mathsf{in}}], \boldsymbol{z}^i[V_{i,j}]) \\ &= \boldsymbol{Q}_{\mathsf{out},\ell}^{(i,j)}((\boldsymbol{w}[V_{i,j}^{\mathsf{in}}], \boldsymbol{z}^i[V_{i,j}]) - (\boldsymbol{r}[V_{i,j}^{\mathsf{in}}], \boldsymbol{r}^i[V_{i,j}])) \\ &= \boldsymbol{Q}_{\mathsf{out},\ell}^{(i,j)}(\boldsymbol{x}[V_{i,j}^{\mathsf{in}}], \boldsymbol{u}^i[V_{i,j}])).\end{aligned}$$

Therefore, the $\boldsymbol{v}_\ell$ are uniformly random conditioned on:

$$\sum_{\ell \in H} \boldsymbol{v}_\ell^{(i,j)} = \boldsymbol{Q}_{\mathsf{out}}^{(i,j)}(\boldsymbol{x}[V_{i,j}^{\mathsf{in}}], \boldsymbol{u}^i[V_{i,j}]) - \sum_{\ell \in D} \boldsymbol{v}_\ell^{(i,j)}.$$

By definition of the $\boldsymbol{Q}_{\mathsf{out}}^{(i,j)}(\boldsymbol{X})$, we have $\boldsymbol{Q}_{\mathsf{out}}^{(i,j)}(\boldsymbol{x}[V_{i,j}^{\mathsf{in}}], \boldsymbol{u}^i[V_{i,j}]) = \boldsymbol{v}^{(i,j)}$, where the $\boldsymbol{v}^{(i,j)}$ are such that $f_i(\boldsymbol{x}, \boldsymbol{u}^i) = (\boldsymbol{u}^{i+1}, (\boldsymbol{v}^{(i,j)})_{1 \le j \le \lceil m_i/\beta \rceil})$. Therefore, by definition of the f_i, the vectors $\boldsymbol{v}_\ell$ are uniformly random conditioned on:

$$\sum_{\ell \in H} \boldsymbol{v}_\ell = \boldsymbol{v} - \sum_{\ell \in D} \boldsymbol{v}_\ell, \text{ where } \boldsymbol{v} = C(\boldsymbol{x}).$$

This concludes the proof that Sim perfectly simulates Π_C. □

5 Generating Correlated Randomness from LPN

In this section, we construct a protocol Π_{corr}, which implements the ideal functionality $\mathscr{F}_{\mathsf{corr}}$ with small communication, under the quasi-polynomial LPN assumption. A very natural approach to realise a functionality that distributes correlated random coins using a small amount of communication is to rely on *pseudorandom correlation generators*, a primitive recently defined an constructed (for various types of correlations, and under a variety of assumptions) in [BCG+19b]. At a high level, [BCG+19b] suggests to distribute correlated randomness with the following approach:

– Use a generic secure computation protocol Π_{Gen} to distributively execute the $\mathsf{PCG.Gen}$ functionality of the pseudorandom correlation generator. Note that $\mathsf{PCG.Gen}$ outputs short seeds, much smaller than the correlated pseudo-random strings which can be stretched from these seeds. Therefore, Π_{Gen} can potentially have a relatively high communication overhead in its inputs and outputs, while maintaining the overall communication overhead of Π_{corr} small.
– Expand the distributively generated seeds locally using the Expand algorithm of the PCG. Each such string is guaranteed, by the security of the PCG, to be indistinguishable (from the viewpoint of the other parties) from a uniformly random string sampled conditioned on satisfying the target correlation with the expanded strings held by the other parties.

While this approach does not necessarily leads to a secure implementation of an ideal functionality generating correlated random coins, it was shown in [BCG+19b] (Theorem 19 in [BCG+19b]) that it provides a provably secure implementation for all *corruptible* ideal functionalities for distributing correlated random coins. Note that this property is satisfied by our functionality $\mathscr{F}_{\mathsf{corr}}$. Our protocol Π_{corr} will follow this approach. We start by constructing a pseudorandom correlation generator for the type of correlated randomness produced by $\mathscr{F}_{\mathsf{corr}}$, building upon an LPN-based construction of [BCG+19b].

5.1 Substrings Tensor Powers Correlations (stp)

We now describe our construction of a PCG for generating the type of correlated randomness produced by $\mathscr{F}_{\mathsf{corr}}$. As all constructions of [BCG+19b], our construction will be restricted to the two-party setting; hence, we focus on $N = 2$ parties from now on. Abstracting out the unnecessary details, the functionality $\mathscr{F}_{\mathsf{corr}}$ does the following. It is parametrised with a vector length w, subsets $(S_i)_{1\le i\le \mathsf{n_s}} \in \binom{[w]}{\le K}^{\mathsf{n_s}}$, a tensor power parameter tpp, and generates shares of:

$$(\boldsymbol{r}, ((1_{\mathbb{F}} \mid\mid \boldsymbol{r}[S_i])^{\otimes \mathsf{tpp}})_{1\le i\le \mathsf{n_s}}), \text{ where } \boldsymbol{r} \in \mathbb{F}^w \text{ is random.}$$

We call $\mathscr{C}$ the correlation generator associated with $\mathscr{F}_{\mathsf{corr}}$, i.e. the PPT algorithm that, on input the security parameter in unary 1^λ, samples correlated random string as above (where the parameters $(\mathsf{n_s}, K, \mathsf{tpp})$ are functions of λ). It is straightforward to see that $\mathscr{C}$ is a reverse-samplable correlation generator (see Definition 9), since it is an additive correlation: given any fixed share share_0, a matching share can be reverse-sampled by sampling $\boldsymbol{r}$ and setting $\mathsf{share}_1 \leftarrow (\boldsymbol{r}, ((1_{\mathbb{F}} \mid\mid \boldsymbol{r}[S_i])^{\otimes \mathsf{tpp}})_{1\le i\le \mathsf{n_s}}) - \mathsf{share}_0$. We call this type of correlated randomness a *subsets tensor powers* (stp). Below, we describe a pseudorandom correlation generator for such correlations.

5.2 Good Cover

Before we proceed with the description of a PCG to generate such correlations, we need to introduce a concept, that of a *good cover*. The notations in this subsection are completely self-contained, and may conflict with the parameters defined for the main protocol. In the course of our construction we will want to solve the following problem: given a vector $\boldsymbol{v}$ of size n, a family $(S_i)_{i\in[t]} \in \mathscr{P}([n])^t$ of t (*short*) subsets of coordinates of $\boldsymbol{v}$, and a (*small*) bound $B > 0$, the problem is to find a family $(\boldsymbol{v}_j)_{j\in[M]}$ of some number m of size-K subvectors of $\boldsymbol{v}$ such that:

1. The subvectors collectively cover $\boldsymbol{v}$;
2. For each $i \in [t]$, there are at most B subvectors in $(\boldsymbol{v}_j)_{j\in[M]}$ whose coordinates intersect S_i.

We call such a family a *B-Good Cover* of $(\boldsymbol{v}, (S_i)_{i\in[t]})$. First of all we note that the values of the vectors and subvectors do not matter, so we will conflate them with sets and subsets (of coordinates) for simplicity, which leads to a more natural formulation.

Definition 13 (Good Cover – Set Formulation). *Let $n, B, K, t, q, M \in \mathbb{N}$ and $(S_i)_{i\in[t]} \in \binom{[n]}{\le q}^t$ a family of t subsets of $[n]$ of size at most q each. A family $A = (\boldsymbol{\alpha}^j)_{j\in[M]} \in \binom{[n]}{K}^M$ is a B-*Good Cover *of $(S_i)_{i\in[t]}$ if:*

1. *A* covers *$[n]$: $\bigcup_{j=1}^M \boldsymbol{\alpha}^j = [n]$*
2. *Each S_i intersects at most B elements of A: $\forall i \in [t], |\{j \in [M] : \boldsymbol{\alpha}^j \cap S_i \neq \emptyset\}| \le B$.*

We abusively conflate the two views, where a good cover is just a family of subsets $A \in \binom{[n]}{K}^M$ and where the good cover is a family of sparse vectors—given by a set of coordinates and a short vector of values—$A \in (\binom{[n]}{K} \times \mathbb{F}^K)^M$.

Lemma 14 (Random Covers are Good Covers.). *Let $n, \kappa, \kappa' \in \mathbb{N} \setminus \{0, 1\}$, and $(S_i)_{i\in[t]} \in \binom{[n]}{\le q}^t$ a family of t subsets of $[n]$ of size at most q each. Let $A = (\boldsymbol{\alpha}^j)_{j\in[M]} \in \binom{[n]}{K}^M$ be a sequence of M* i.i.d. *uniform random size-K subsets of $[n]$, with $M = \kappa \cdot n \ln n / K$. Let $B \leftarrow \kappa' \kappa \cdot q \cdot \ln n$.*

It holds that $A = (\boldsymbol{\alpha}^j)_{j\in[M]}$ is a B-Good Cover of $(S_i)_{i\in[t]}$ with probability at least:

$$1 - \frac{1}{n^{\kappa-1}} - \frac{t}{n^{(\kappa'-2)\kappa\cdot q/2}}.$$

The proof, which is obtained in a straightforward fashion by combining the union and Chernoff bounds, is given in

Proof.

1. The probability that $i \notin A$ (for any $i \in [n]$) is equal to $(1 - \binom{n-1}{K-1}/\binom{n}{K})^M = (1 - \frac{K}{n})^M = e^{M\cdot\ln(1-K/n)} \le e^{-MK/n} = n^{-\kappa}$ (using $\forall x \ge -1, \ln(1+x) \le x$), so by union bound, the probability that A does not cover $[n]$ is at most $n \cdot n^{-\kappa}$.

2. For each $i \in [q]$, we arbitrarily extend S_i to an $\tilde{S}_i$ of size exactly q; in particular $\forall i \in [q], S_i \subseteq \tilde{S}_i$ so $|\{j : \boldsymbol{\alpha}^j \cap S_i \neq \emptyset\}| \leq |\{j : \boldsymbol{\alpha}^j \cap \tilde{S}_i \neq \emptyset\}|$. For any $j \in [M]$, the indicator random variable of the event $\{\boldsymbol{\alpha}^j \cap \tilde{S}_i \neq \emptyset\}$ follows a Bernouilli law with parameters $(M, \frac{q}{p})$, where $p = 1 - \binom{n-q}{K}/\binom{n}{K}$. Its expectancy is $\mu := \mathbb{E}(X) = M \cdot (1 - \binom{n-q}{K}/\binom{n}{K})$. Note that, using Bernouilli's inequality ($\forall r \in \mathbb{N}^*, \forall x \geq -1, (1+x)^r > (1+rx)$):

$$\mu = M \cdot \left(1 - \frac{\prod_{j=0}^{K-1}(n-q-j)}{\prod_{j=0}^{K-1}(n-j)}\right) \leq m \cdot (1 - \frac{(n-2q)^K}{n^K})$$
$$= M \cdot (1 - (1 - 2q/n)^K) \leq M\frac{2qK}{n} = 2\kappa \cdot q \cdot \ln n.$$

Therefore the probability that $X := |\{j : \alpha_j \in \tilde{S}_i\}| \leq B$ is, by (a looser version of the multiplicative form of) Chernoff's bound, at most:

$$\forall \delta \geq 0, \Pr[X > (1+\delta)\mu] \leq \exp\left(-\frac{\delta^2\mu}{2+\delta}\right).$$

Setting $\delta \leftarrow B/\mu - 1$ yields:

$$\Pr[X > B] \leq \exp\left(-\frac{\delta^2\mu}{2+\delta}\right) \leq \exp(-\delta\mu/2) = \exp\left(-\frac{B-\mu}{2}\right)$$
$$\leq n^{-(\kappa'-2)\kappa \cdot q/2}.$$

3. The desired bound is then obtained by union bound.

$\square$

5.3 PCG for Subsets Tensor Powers ($\mathsf{PCG_{stp}}$)

We now proceed with the description of a pseudorandom correlation generator for subsets tensor powers.

PCG for Low-Degree Polynomials from [BCG+19b]. We start by recalling a natural variant of pseudorandom correlation generator of [BCG+19b, Section 6], which generates shares of $\boldsymbol{r}^{\otimes \mathsf{tpp}}$, for a parameter tpp and a pseudorandom $\boldsymbol{r}$. It relies on the xLPN assumption with dimension n, number of samples $n' > n$, and a number λ of noisy coordinates. In our instantiation, we will typically consider $n' = O(n)$, e.g. $n' = 12n$; this corresponds to a particularly conservative variant of LPN with a very limited number of samples, and is equivalent to the hardness of decoding a random constant-rate linear code (which is known as the *syndrome decoding* problem). As discussed in Section 3, all known attacks on the syndrome decoding problem for constant-rate codes have complexity $2^{O(\lambda)}$. The PCG of [BCG+19b] is parametrised by integers $1^\lambda, n, n', \lambda, \mathsf{tpp} \in \mathbb{N}$ (where $n' > n$), a field $\mathbb{F}$, and a random parity-check matrix $H_{n',n} \xleftarrow{\$} \mathbb{F}^{(n'-n)\times n'}$.

PCG for Degree-tpp Polynomial Correlations

PCG.Gen: On input 1^λ:

1. Pick a random λ-sparse vector $\boldsymbol{e} \xleftarrow{\$} \mathsf{HW}_\lambda(\mathbb{F}^{n'})$. Note that $\boldsymbol{e}^{\otimes \mathsf{tpp}} \in \mathsf{HW}_{\lambda^{\mathsf{tpp}}}(\mathbb{F}^{(n')^{\mathsf{tpp}}})$. Let $f : [(n')^{\mathsf{tpp}}] \mapsto \mathbb{F}$ be the multi-point function with λ^{tpp} points, such that $f(i)$ returns the i-th coordinate of $\boldsymbol{e}^{\otimes \mathsf{tpp}}$.
2. Compute $(K_0^{\mathsf{fss}}, K_1^{\mathsf{fss}}) \xleftarrow{\$} \mathsf{MPFSS.Gen}(1^\lambda, f)$. Output $\mathsf{k}_0 \leftarrow (n, K_0^{\mathsf{fss}})$ and $\mathsf{k}_1 \leftarrow (n, K_1^{\mathsf{fss}})$.

PCG.Expand: On input $(\sigma, \mathsf{k}_\sigma)$, compute $\boldsymbol{v}_\sigma \leftarrow \mathsf{MPFSS.FullEval}(\sigma, K_\sigma^{\mathsf{fss}})$ in $\mathbb{F}^{(n')^{\mathsf{tpp}}}$ and set $\boldsymbol{r}_\sigma \leftarrow H_{n',n}^{\otimes \mathsf{tpp}} \cdot \boldsymbol{v}_\sigma$. Output $\boldsymbol{r}_\sigma$.

Fig. 6: PCG for Low-Degree Polynomials from [BCG+19b].

Correctness follows from the fact that $\boldsymbol{v}_0 + \boldsymbol{v}_1 = \boldsymbol{e}^{\otimes \mathsf{tpp}}$ by the correctness of MPFSS, and $H_{n',n}^{\otimes \mathsf{tpp}} \cdot \boldsymbol{e}^{\otimes \mathsf{tpp}} = (H_{n',n} \cdot \boldsymbol{e})^{\otimes \mathsf{tpp}}$ by multilinearity of the tensor product. Hence, denoting $\boldsymbol{r} = H_{n',n} \cdot \boldsymbol{e}$, it holds that $\boldsymbol{r}_0 + \boldsymbol{r}_1 = \boldsymbol{r}^{\otimes \mathsf{tpp}}$. For security, we must show that the following distributions are indistinguishable for any $\sigma = 0, 1$:

$$\begin{aligned}
&\{(\mathsf{k}_\sigma, \boldsymbol{r}_{1-\sigma}) \ : \ (\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{Gen}(1^\lambda), \boldsymbol{r}_{1-\sigma} \leftarrow \mathsf{Expand}(1-\sigma, \mathsf{k}_{1-\sigma})\} \\
\overset{c}{\approx}&\{(\mathsf{k}_\sigma, \boldsymbol{r}_{1-\sigma}) \ : \ (\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{Gen}(1^\lambda), \boldsymbol{r}_{\sigma} \leftarrow \mathsf{Expand}(\sigma, \mathsf{k}_{\sigma}), \boldsymbol{r} \xleftarrow{\$} \mathbb{F}^n, \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \boldsymbol{r}_{1-\sigma} \leftarrow \boldsymbol{r}^{\otimes \mathsf{tpp}} - \boldsymbol{r}_\sigma\}
\end{aligned}$$

Proof. We sketch the analysis for the sake of completeness; the full proof is given in [BCG+19b]. Security is shown with the following sequence of hybrids: first generate $(\mathsf{k}_\sigma, \boldsymbol{r}_{1-\sigma})$ as in the first distribution above. Then, generate $(\mathsf{k}_\sigma, \boldsymbol{r}_{1-\sigma})$ as before, and generate an alternative key k'_σ solely from the parameters $(1^\lambda, \mathbb{F}, n, n', t, \mathsf{tpp})$, using the simulator of the MPFSS. Output $(\mathsf{k}'_\sigma, \boldsymbol{r}_{1-\sigma})$; under the security of the MPFSS, this distribution is indistinguishable from the previous one. Note that k'_σ does not depend anymore on the noise vector $\boldsymbol{e}$. In the next hybrid, generate $\boldsymbol{r} \xleftarrow{\$} H_{n',n} \cdot \boldsymbol{e}$ and set $\boldsymbol{r}_{1-\sigma} \leftarrow \boldsymbol{r}^{\otimes \mathsf{tpp}} - \mathsf{Expand}(\sigma, \mathsf{k}_\sigma)$; this game is perfectly indistinguishable from the previous one. Finally, replace $\boldsymbol{r} \xleftarrow{\$} H_{n',n} \cdot \boldsymbol{e}$ by $\boldsymbol{r} \xleftarrow{\$} \mathbb{F}^n$; under the LPN assumption, this last game (which correspond exactly to the second distribution) is computationally indistinguishable from the previous one, and security follows. $\square$

Our New PCG. We now describe a variant of the above PCG, tailored to computing the tensor powers of many short subsets. The PCG is parametrised by $(S_i)_{i \in [K]} \in \binom{[w]}{\leq K}^{\mathsf{n_s}}$, $\mathsf{n_s}$ subsets of at most K indices taken from $[w]$. We assume for simplicity, but morally without loss of generality[6], that $\bigcup_{i=1}^{\mathsf{n_s}} S_i = [w]$. Our goal is for the parties to obtain shares of some pseudorandom vector $\boldsymbol{r} \in \mathbb{F}^w$ as well as shares of $(1 \mid\mid \boldsymbol{r}[S_i])^{\otimes \mathsf{tpp}} \in \mathbb{F}^{w \cdot \mathsf{tpp}}$ for each $i \in [\mathsf{n_s}]$.

We start by generating a B-good cover (for some integer B) of the $(S_i)_i$ of the form $(\alpha_j, \boldsymbol{r}_j)_{j \in [m]} \in (\binom{[w]}{\theta} \times \mathbb{F}^\theta)^m$ where each $\boldsymbol{r}_j$ is pseudorandom. We generate each of the m pseudorandom masks $\boldsymbol{r}_j$ using a different instance of xLPN, *i.e.* $\boldsymbol{r}_j \leftarrow H_j \cdot \boldsymbol{e}_j$, where $\boldsymbol{e}_j \in \mathbb{F}^{\theta'}$ is λ-sparse and $H_j \xleftarrow{\$} \mathbb{F}^{\theta \times \theta'}$ for some $\theta' = O(\theta)$. For each S_i, we denote $I_i \coloneqq \{j \in [m] \colon \alpha_j \cap S_i \neq \emptyset\} = \{j_1, \ldots, j_{|I_i|}\}$ the set of the indices of the masks which 'intersect' with S_i. Note that $\forall i \in [\mathsf{n_s}], |I_i| \leq B$ by definition of a B-good cover. We can now proceed with our main goal: generating shares of a subsets tensor powers correlation.

We define $\boldsymbol{r} \coloneqq \sum_{j=1}^m f_{\alpha_j, \boldsymbol{r}_j} \in \mathbb{F}^w$, where $f_{\alpha_j, \boldsymbol{r}_j} \in \mathbb{F}^w$ is the sparse vector defined by $(f_{\alpha_j, \boldsymbol{r}_j})_{|\alpha_j} = \boldsymbol{r}_j$ (and which is equal to $0_\mathbb{F}$ on $[w] \smallsetminus \alpha_j$). Since $\bigcup_{i=1}^{\mathsf{n_s}} S_i = [w]$ and each of the $\boldsymbol{r}_j$ is pseudorandom, $\boldsymbol{r}$ is also pseudorandom.

Note that for any given $i \in [\mathsf{n_s}]$, $(1_\mathbb{F} \mid\mid \boldsymbol{r}[S_i])$ is a subvector of the vector $\tilde{\boldsymbol{r}}_i$ obtained by multiplying the block-diagonal matrix $H'_i = \mathsf{Diag}((1_\mathbb{F}), H_{j_1}, \ldots, H_{j_{|I_i|}})$ with the vector $\boldsymbol{e}'_i = (1_\mathbb{F} || e_{j_1} || \cdots || e_{j_{|I_i|}})$. Therefore for any tensor power tpp (*i.e.* the degree of the polynomial correlation), $\tilde{\boldsymbol{r}}_i^{\otimes \mathsf{tpp}} = (H'_i \cdot \boldsymbol{e}'_i)^{\otimes \mathsf{tpp}} = (H'_i)^{\otimes \mathsf{tpp}} \cdot (\boldsymbol{e}'_i)^{\otimes \mathsf{tpp}}$. If the parties use an MPFSS scheme to generate small seeds which expand to $(\boldsymbol{e}'_i)^{\otimes \mathsf{tpp}}$, they can then locally obtain shares of $\tilde{\boldsymbol{r}}_i^{\otimes \mathsf{tpp}}$ (since $(H'_i)^{\otimes \mathsf{tpp}}$ is public), and therefore of $(1_\mathbb{F} \mid\mid \boldsymbol{r}[S_i])^{\otimes \mathsf{tpp}}$. From all these shares of all the $(1_\mathbb{F} \mid\mid \boldsymbol{r}[S_i])^{\otimes \mathsf{tpp}}, i \in [\mathsf{n_s}]$ the parties can locally extract shares of all the $\boldsymbol{r}[S_i]$ and thence shares of $\boldsymbol{r}$ (since $\bigcup_{i=1}^{\mathsf{n_s}} S_i = [w]$). The protocol is given in Figure 7.

Theorem 15. *Let $w > 0$, and $(S_i)_{i \in [\mathsf{n_s}]}$ a list of $\mathsf{n_s}$ subsets of $[w]$. Let B, θ' such that there exists a B-good cover of $(S_i)_{i \in [\mathsf{n_s}]}$ comprised of size-θ' vectors, and let $\theta < \theta'$. Assume that the $\mathbb{F}$-$\mathsf{xLPN}(\theta, \theta', \lambda)$ assumption holds, and that MPFSS is a secure multi-point function secret-sharing scheme for the family of $(1 + \mu \cdot \lambda)^{\mathsf{tpp}}$-point functions from $[(1 + \mu \cdot \theta')^{\mathsf{tpp}}]$ to $\mathbb{F}$ for all $\mu \in [B]$. Then $\mathsf{PCG_{stp}}$ is a secure pseudorandom correlation generator, which generates pseudorandom shares of a subsets tensor powers correlation $(\boldsymbol{r}, ((1_\mathbb{F} \mid\mid \boldsymbol{r}[S_i])^{\otimes \mathsf{tpp}})_{1 \leq i \leq \mathsf{n_s}})$ where $\boldsymbol{r} \in \mathbb{F}^w$.*

[6] If $\bigcup_{i=1}^{\mathsf{n_s}} S_i \neq \emptyset$, and with the notations of the rest of the section, the vector $\boldsymbol{r}$ we generate is equal to $0_\mathbb{F}$ on $[w] \smallsetminus \bigcup_{i=1}^{\mathsf{n_s}} S_i$, hence not pseudorandom. However, we can simply have the parties generate another mask $\boldsymbol{r}' = H' \cdot \boldsymbol{e}'$, pseudorandom under xLPN, to cover $[w] \smallsetminus \bigcup_{i=1}^{\mathsf{n_s}} S_i$. Since the parties do not need shares of $(\boldsymbol{r}')^{\otimes \mathsf{tpp}}$, the communication complexity of generating the λ-sparse $\boldsymbol{e}'$ using an MPFSS is not an issue.

Pseudorandom Correlation Generator $\mathsf{PCG}_{\mathsf{stp}}$

Parameters: $w, \mathsf{tpp}, \lambda \in \mathbb{N}$ and $(S_i)_{1 \leq i \leq \mathsf{n_s}} \subseteq [w]^{\mathsf{n_s}}$.

Gen: On input 1^λ:

1. Generate a family of subsets $(\alpha_j)_{1 \leq j \leq m} \in \binom{[m]}{\theta'}^m$ which form a B-good cover of the $(S_i)_{i \in [\mathsf{n_s}]}$ (when the α_j are paired with length-θ' vectors in $\mathbb{F}^{\theta'}$), and contracting matrices[a] $(H_j)_{j \in [m]} \in (\mathbb{F}^{\theta \times \theta'})^m$.
2. Pick m random λ-sparse vectors $\boldsymbol{e}_j \xleftarrow{\$} \mathsf{HW}_\lambda(\mathbb{F}^{\theta'}), j \in [m]$ and define:
$$\boldsymbol{r}_j \leftarrow H_j \cdot \boldsymbol{e}_j^\intercal, \text{ for all } j \in [m].$$
3. For each $i = 1 \ldots \mathsf{n_s}$:
   (a) Denoting $I_i \coloneqq \{j \in [m] \colon \alpha_j \cap S_i \neq \emptyset\} = \{j_1, \cdots, j_{m_i}\}$ (with $m_i \leq B$), set:
   $$\tilde{\boldsymbol{r}}_i \leftarrow (1_\mathbb{F} \mid\mid H_{j_1} \cdot \boldsymbol{e}_{j_1}^\intercal \mid\mid \cdots \mid\mid H_{j_{m_i}} \cdot \boldsymbol{e}_{j_{m_i}}^\intercal)^\intercal.$$
   (b) Let $f_i : [(1 + m_i \cdot \theta')^{\mathsf{tpp}}] \to \mathbb{F}$ be the multi-point function with $(1 + m_i \cdot \lambda)^{\mathsf{tpp}}$ points, such that $f_i(x) = (1_\mathbb{F} || \boldsymbol{e}_{j_1} || \cdots || \boldsymbol{e}_{j_{m_i}})^{\otimes \mathsf{tpp}}[x]$. Compute $(K_{i,0}^{\mathsf{fss}}, K_{i,1}^{\mathsf{fss}}) \xleftarrow{\$} \mathsf{MPFSS.Gen}(1^\lambda, f_i)$.
4. Output $\mathsf{k}_0 \leftarrow (w, (K_{i,0}^{\mathsf{fss}})_{i \leq \mathsf{n_s}})$ and $\mathsf{k}_1 \leftarrow (w, (K_{i,1}^{\mathsf{fss}})_{i \leq \mathsf{n_s}})$.

Expand: On input $(\sigma, \mathsf{k}_\sigma)$, parse k_σ as $(w, (K_{i,\sigma}^{\mathsf{fss}})_{i \leq \mathsf{n_s}})$.

1. For each $i = 1 \ldots \mathsf{n_s}$:
   Set $H_i' \leftarrow \mathsf{Diag}((1_\mathbb{F}), H_{j_1}, \ldots, H_{j_{m_i}})$, compute
   $$\boldsymbol{v}_{i,\sigma} \leftarrow \mathsf{MPFSS.FullEval}(\sigma, K_{i,\sigma}^{\mathsf{fss}}) \in \mathbb{F}^{(1+m_i\lambda)^{\mathsf{tpp}}}$$
   and set $\boldsymbol{y}_\sigma \leftarrow ((H_i')^{\otimes \mathsf{tpp}} \cdot \boldsymbol{v}_\sigma)_{1 \leq i \leq \mathsf{n_s}}$.
2. Extract from $\boldsymbol{y}_\sigma$ the appropriate linear combinations of its elements corresponding to a share of $(\boldsymbol{r}, ((1_\mathbb{F} \mid\mid \boldsymbol{r}[S_i])^{\otimes tpp})_{i \in [\mathsf{n_s}]})$. // If there are several ways to do so, it must be consistent accross $\sigma \in \{0, 1\}$.

[a] Implicitly, the H_j are supposed to be 'suitably chosen' for xLPN to be presumed hard, e.g. that they were randomly and independently sampled.

Fig. 7: Pseudorandom correlation generator $\mathsf{PCG}_{\mathsf{stp}}$ for generating pseudorandom instances of the subsets tensor powers correlation over a field $\mathbb{F}$.

– Communication: *If the* MPFSS *seeds have size* $O[\lambda\cdot(1+B\lambda)^{\mathsf{tpp}}\cdot\log((1+B\theta')^{\mathsf{tpp}})]$ *and* $\mathsf{MPFSS.FullEval}$ *can be computed with* $O((1+B\lambda)^{\mathsf{tpp}}\cdot(1+B\theta')^{\mathsf{tpp}}\cdot\frac{\log|\mathbb{F}|}{\lambda})$ *invocations of a pseudorandom generator* $\mathsf{PRG}:\{0,1\}^\lambda\mapsto\{0,1\}^{2\lambda+2}$*, then* $\mathsf{PCG_{stp}.Gen}$ *outputs seeds of size:*

$$|\mathsf{k}_\sigma| = O\left(\mathsf{n_s}\cdot\lambda\cdot(1+B\lambda)^{\mathsf{tpp}}\cdot\log\left((1+B\theta')^{\mathsf{tpp}}\right)\right).$$

– Computation: *The computational complexity of* $\mathsf{PCG_{stp}.Expand}$ *is predominantly that of* $O(\mathsf{n_s}\cdot(1+B\lambda)^{\mathsf{tpp}}\cdot(1+B\theta')\cdot\frac{\log|\mathbb{F}|}{\lambda})$ *invocations of a PRG, plus* $\mathsf{n_s}$ *matrix-vector products with a matrix of dimensions* $(1+B\theta)^{\mathsf{tpp}}\times(1+B\theta')^{\mathsf{tpp}}$ *which requires at most* $O(\mathsf{n_s}\cdot(B\theta)^{\mathsf{tpp}}\cdot(B\theta')^{\mathsf{tpp}})\subseteq O(\mathsf{n_s}\cdot(B\theta')^{2\cdot\mathsf{tpp}})$ *arithmetic operations over* $\mathbb{F}$.

Proof. Let us first show correctness of our candidate PCG. By correctness of the MPFSS, $\boldsymbol{v}_{i,0}+\boldsymbol{v}_{i,1}=(1_{\mathbb{F}}||(\boldsymbol{e}_\ell)_{\ell\in I_i})^\intercal$ for every $i\in[\mathsf{n_s}]$. Therefore,

$$\begin{aligned}\boldsymbol{y}_0+\boldsymbol{y}_1 &= ((H_i')^{\otimes\mathsf{tpp}}\cdot\boldsymbol{v}_0+(H_i')^{\otimes\mathsf{tpp}}\cdot\boldsymbol{v}_1)_{1\le i\le\mathsf{n_s}}\\ &= ((H_i')^{\otimes\mathsf{tpp}}\cdot((1_{\mathbb{F}}\mid\mid(\boldsymbol{e}_\ell)_{\ell\in I_i})^\intercal)_{1\le i\le\mathsf{n_s}} = (\tilde{\boldsymbol{r}}_i{}^{\otimes\mathsf{tpp}})_{1\le i\le\mathsf{n_s}}.\end{aligned}$$

$\boldsymbol{r}[S_i]=\left(\sum_{j=1}^m f_{\alpha_j,\boldsymbol{r}_j}\right)[S_i]=\left(\sum_{j\in I_i} f_{\alpha_j,\boldsymbol{r}_j}\right)[S_i]$, where $f_{\alpha_j,\boldsymbol{r}_j}$ is the sparse vector obtained by spreading the vector $\boldsymbol{r}_j$ over the coordinates in the ordered set α_j. It follows that $(1_{\mathbb{F}}\mid\mid\boldsymbol{r}[S_i])^{\otimes\mathsf{tpp}}$ can be extracted from $\tilde{\boldsymbol{r}}_j$ (since any degree-($\le\mathsf{tpp}$) polynomial in $\boldsymbol{a}+\boldsymbol{b}$ is also a degree-($\le\mathsf{tpp}$) polynomial in $\boldsymbol{a}||\boldsymbol{b}$).

Security follows exactly by the same sequence of hybrids as in the previous analysis. We show the following indistinguishability:

$$\begin{aligned}&\{(\mathsf{k}_\sigma,\boldsymbol{r}_{1-\sigma})\ :\ (\mathsf{k}_0,\mathsf{k}_1)\xleftarrow{\$}\mathsf{Gen}(1^\lambda),\boldsymbol{r}_{1-\sigma}\leftarrow\mathsf{Expand}(1-\sigma,\mathsf{k}_{1-\sigma})\}\\ \overset{c}{\approx}&\{(\mathsf{k}_\sigma,\boldsymbol{r}_{1-\sigma})\ :\ (\mathsf{k}_0,\mathsf{k}_1)\xleftarrow{\$}\mathsf{Gen}(1^\lambda),\boldsymbol{r}_\sigma\leftarrow\mathsf{Expand}(\sigma,\mathsf{k}_\sigma),\\ &\qquad\qquad\boldsymbol{r}'\xleftarrow{\$}\mathbb{F}^m,\boldsymbol{r}_{1-\sigma}\leftarrow((1_{\mathbb{F}}||\boldsymbol{r}')^{\otimes\mathsf{tpp}})_{i\le\mathsf{n_s}}-\boldsymbol{r}_\sigma\}.\end{aligned}$$

We first switch all the $K^{\mathsf{fss}}_{i,\sigma}$ to simulated keys using the security of the MPFSS, then replace the $H_i\cdot\boldsymbol{e}_i^\intercal$ by random vectors, applying m times the security of LPN, once for each replacement. The sum of sparse random vectors which form a good cover is itself a random vector (by item 1 in definition 13), therefore the resulting distribution is exactly the second distribution above, hence security follows.

Finally, the efficiency claims can be read directly from the construction, and follows from the fact that Gen consists of $\mathsf{n_s}$ MPFSS seeds, and the cost of Expand is dominated by $\mathsf{n_s}$ calls to $\mathsf{MPFSS.FullEval}$ and $\mathsf{n_s}$ matrix-vector products. □

5.4 Instantiating the MPFSS

Theorem 15 assumes the existence of an MPFSS scheme MPFSS for the family of all $(1+\mu\cdot\lambda)^{\mathsf{tpp}}$-point functions from $[(1+\mu\cdot\theta')^{\mathsf{tpp}}]$ to $\mathbb{F}$ for some $\mu\in[B]$ (or, equivalently, an MPFSS for each μ which can then all be combined into one scheme), with the following efficiency guarantees: $\mathsf{MPFSS.Gen}(1^\lambda)$ outputs seeds of size $O((1+B\lambda)^{\mathsf{tpp}}\cdot\lambda\cdot\log((1+B\theta')^{\mathsf{tpp}}))$, and $\mathsf{MPFSS.FullEval}$ can be computed with $O((1+B\lambda)^{\mathsf{tpp}}\cdot(1+B\theta')^{\mathsf{tpp}}\cdot\frac{\log|\mathbb{F}|}{\lambda})$ invocations of a pseudorandom generator $\mathsf{PRG}:\{0,1\}^\lambda\mapsto\{0,1\}^{2\lambda+2}$. The works of [BGI16b, BCGI18] provides exactly such a construction, which makes a black box use of any pseudorandom generator $\mathsf{PRG}:\{0,1\}^\lambda\mapsto\{0,1\}^{2\lambda+2}$.

We instantiate the PRG using the LPN-based construction of [BKW03], which we now recall. Fix some (constant) noise rate ε, a random matrix $A\xleftarrow{\$}\mathbb{F}_2^{n_1\times n_2}$. Given a random bitstring $r\in\{0,1\}^{n_2+n_1\cdot h(\varepsilon)}$, where h is the binary entropy function, define $\boldsymbol{s}=\boldsymbol{s}(r)\in\mathbb{F}_2^{n_2}$ to be the n_2 first bits of r, and use the remaining $n_1\cdot h(\varepsilon)$ bits to sample a random vector $\boldsymbol{e}(r)$ from $\mathsf{Ber}_\varepsilon(\mathbb{F}_2)^{n_1}$ (it is well known that this distribution can be sampled using roughly $n_1\cdot h(\varepsilon)$ bits of randomness). Define the pseudorandom generator $\mathsf{PRG}:\{0,1\}^\lambda\mapsto\{0,1\}^{2\lambda+2}$ as $\mathsf{PRG}(r)=A\cdot\boldsymbol{s}(r)+\boldsymbol{e}(r)\in\mathbb{F}_2^{n_1}$. Security follows from the $\mathbb{F}_2$-$\mathsf{LPN}(n_2,n_1,\varepsilon)$ assumption, and this PRG stretches $\lambda=n_2+n_1\cdot h(\varepsilon)$ bits to $n_1=2\lambda+2$ bits when $n_2=n_1\cdot(1/2-h(\varepsilon))-1$. Hence, given the security parameter λ, security follows from the $\mathbb{F}_2$-$\mathsf{LPN}(\lambda\cdot(1-2h(\varepsilon))-2h(\varepsilon),2\lambda+2,\varepsilon)$ assumption for any constant ε; for example, setting $\varepsilon=1/8$, this assumption is implied by the $\mathbb{F}_2$-$\mathsf{LPN}(\lambda/4,3\lambda,1/8)$ assumption. The cost of evaluating PRG is dominated by the matrix-vector product $A\cdot\boldsymbol{s}$, which requires at most $n_1n_2=O(\lambda^2)$ arithmetic operations.

5.5 Securely Distributing $\mathsf{MPFSS.Gen}$ and Π_{stp}

The seeds of the MPFSS scheme of [BCGI18] can be securely generated by using parallel instances of a generic secure computation protocols to securely evaluate the above PRG. Using GMW to instantiate the generic protocol, we have:

Corollary 16. *There exists a semi-honest secure two-party protocol Π_{MPFSS} which distributes the seeds of a multi-point function secret-sharing scheme* MPFSS *for the family of t'-point functions from $[(1+B\theta')^{\mathsf{tpp}}]$ to $\mathbb{F}$, using $O(t'\cdot\nu\cdot\lambda^2)$ calls to an ideal oblivious transfer functionality, where $\nu=\log((1+B\theta')^{\mathsf{tpp}})$ and $t'=(1+B\lambda')^{\mathsf{tpp}}$, with an additional communication of $O(t'\cdot\nu\cdot\lambda^2)$ bits, and total computation polynomial in $t'\cdot\nu\cdot\lambda$.*

Proof. Let $t'\leftarrow(1+B\lambda)^{\mathsf{tpp}}$. The MPFSS scheme MPFSS for the family of t'-point functions from $[(1+B\theta')^{\mathsf{tpp}}]$ to $\mathbb{F}$ of [BCGI18] is constructed using t' independent instances of a single-point function secret sharing scheme (also called *distributed point function* [GI14]): any t'-point function $f:[(1+B\theta')^{\mathsf{tpp}}]\mapsto\mathbb{F}$ can be written as the sum $\sum_{i=1}^{t'}f_i$ of point functions $f_i:[(1+B\theta')^{\mathsf{tpp}}]\mapsto\mathbb{F}$ which evaluate to $0_{\mathbb{F}}$ everwhere, except on a single entry j_i where they take the value $f(j_i)$ (the j_i being the entries on which f does not evaluate to $0_{\mathbb{F}}$). Given a distributed point function $(\mathsf{Gen},\mathsf{Eval})$, the construction and its analysis are straightforward:

- $\mathsf{MPFSS.Gen}$: On input $(1^\lambda,f)$, decompose f as $\sum_{i=1}^{t'}f_i$ as above, and output $(\mathsf{Gen}(1^\lambda,f_i))_{1\le i\le t'}$.
- $\mathsf{MPFSS.Eval}$: On input $(\sigma,(K_{i,\sigma})_{i\in[t']},x)$, output $y_\sigma\leftarrow\sum_{i=1}^{t'}\mathsf{Eval}(K_{i,\sigma},f_i,x)$.

Therefore, securely distributing MPFSS seeds reduces to t' invocations of a secure protocol for distributing the seeds of a distributed point function (DPF). The $\mathsf{DPF.Gen}$ construction of [BGI16b] for point functions over the domain $[(1+B\theta')^{\mathsf{tpp}}]$ works as follows: the two output keys are $K_0=(s_0^{(0)},cw_1,\dots,cw_{\nu+1})$ and $K_1=(s_1^{(0)},cw_1,\dots,cw_{\nu+1})$ where $s_0^{(0)},s_1^{(0)}$ are two random seeds for the PRG and $\nu=\log((1+B\theta')^{\mathsf{tpp}})$. Gen proceeds in $\nu+1$ steps. In the i-th step it expands $s_0^{(i-1)}$ and $s_1^{(i-1)}$ by using one PRG invocation for each seed and obtains $s_0^{(i)},s_1^{(i)}$, and cw_i. In the final step the algorithm computes $cw_{\nu+1}$ as a function of the expanded seeds and the target value.

Securely Generating DPF Seeds. We recall that the well known GMW protocol [GMW87b] allows two parties to securely evaluate any circuit of size s (in the semi-honest model) using $O(s)$ calls to an oblivious transfer functionality, and $O(s)$ additional bits of communication. Using GMW for distributing the Gen procedure of the DPF over a field $\mathbb{F}$, the communication and computation of the protocol are dominated by two factors: $O(\lambda)$ oblivious transfers for a seed and location of the designated point and by $O(\nu)$ secure evaluations of the PRG. Since evaluating the PRG can be done using $O(\lambda^2)$ arithmetic operations over $\mathbb{F}_2$, it can be generically computed using $O(\lambda^2)$ calls to an oblivious transfer functionality, and $O(\lambda^2)$ additional bits of communication. Hence the following lemma:

Lemma 17. *There exists a semi-honest secure two-party protocol Π_{DPF} which distributes the seeds of a distributed point function* DPF *for the family of point functions from $[(1+B\theta')^{\mathsf{tpp}}]$ to $\mathbb{F}$, using $O(\nu\cdot\lambda^2)$ calls to an ideal oblivious transfer functionality, where $\nu=\log((1+B\theta')^{\mathsf{tpp}})$, with an additional communication of $O(\nu\cdot\lambda^2)$ bits, and total computation polynomial in $\nu\cdot\lambda$.*

$\square$

As a direct corollary of Corollary 16, since the seeds of $\mathsf{PCG}_{\mathsf{stp}}$ contain exactly $\mathsf{n_s}$ independent MPFSS seeds, we have:

Corollary 18. *There exists a semi-honest secure two-party protocol Π_{stp} which distributes the seeds of the pseudorandom correlation generator* $\mathsf{PCG}_{\mathsf{stp}}$ *represented on Figure 7, using $O(\mathsf{n_s}\cdot t'\cdot\nu\cdot\lambda^2)$ calls to an ideal oblivious transfer functionality, where $\nu=\log((B\theta'+1)^{\mathsf{tpp}})$ and $t'=(1+B\lambda)^{\mathsf{tpp}}$, with an additional communication of $O(\mathsf{n_s}\cdot t'\cdot\nu\cdot\lambda^2)$ bits, and total computation $O(\mathsf{n_s}\cdot\mathsf{poly}(t'\cdot\nu\cdot\lambda))$.*

Instantiating the oblivious transfer. To execute the GMW protocol, we need an oblivious transfer. Under the $\mathbb{F}_2$-$\mathsf{LPN}(\lambda,O(\lambda),1/\lambda^\delta)$ assumption (δ is any small constant), there exists oblivious transfers (with simulation security) with $\mathsf{poly}(\lambda)$ communication and computation; see for example [DGH+20].

Constructing Π_{corr}. The work of [BCG+19b] shows that any corruptible functionality distributing the output of a correlation generator $\mathscr{C}$ can be secure instantiated using any semi-honest secure two-party protocol Π for distributing the Gen procedure of a PCG for $\mathscr{C}$, with the same communication as Π, and with computational complexity dominated by the computational complexity of Π plus the computational complexity for computing the $\mathsf{PCG.Expand}$ procedure. Therefore, using their result together with our protocol Π_{stp} for generating the seeds of a PCG for subsets tensor powers correlation allows to securely instantiate $\mathscr{F}_{\mathsf{corr}}$ (with $N = 2$).

Recall that the computation of $\mathsf{PCG}_{\mathsf{stp}}.\mathsf{Expand}$ is dominated by $O(\mathsf{n_s}\cdot(1+B\lambda)^{\mathsf{tpp}}\cdot(1+B\theta')^{\mathsf{tpp}}\cdot\frac{\log|\mathbb{F}|}{\lambda})$ invocations of a PRG – which requires at most $O(\lambda^2\cdot\mathsf{n_s}\cdot(1+B\lambda)^{\mathsf{tpp}}\cdot(1+B\theta')^{\mathsf{tpp}}\cdot\frac{\log|\mathbb{F}|}{\lambda})$ operations over $\mathbb{F}_2$ using the simple LPN-based PRG from [BKW03] –, plus an additional $O(\mathsf{n_s}\cdot(1+B\theta)^{\mathsf{tpp}}\cdot(1+B\theta')^{\mathsf{tpp}})$ arithmetic operations over $\mathbb{F}$. Since each operation over $\mathbb{F}$ can be computed with $O(\log|\mathbb{F}|)^2)$ boolean operations, combining the two, we get computation $O(\lambda\cdot\mathsf{n_s}\cdot(1+B\theta)^{\mathsf{tpp}}\cdot(1+B\theta')^{\mathsf{tpp}}\cdot(\log|\mathbb{F}|)^2)$.

All that remains is for the parties to generate the necessary material for $\mathsf{PCG}_{\mathsf{stp}}$: m random $\mathbb{F}^{\theta\times\theta'}$ matrices and m size-θ' subsets of $[w]$. At its core, this is just a matter for the parties to generate and hold the same $m\cdot(\theta\cdot\theta'\cdot\log|\mathbb{F}|+\log\binom{w}{\theta'})$ (pseudo)-random bits. This can be achieved by having one party sample a seed of size λ, send it to the other, and both parties can expand it locally by calling the length-doubling PRG from [BKW03] (and used above) $m\cdot\theta'\cdot(\theta\cdot\log|\mathbb{F}|+\log w)/\lambda$ times (in a GGM tree-like approach). This requires λ bits of communication and $O(m\cdot\theta'\cdot(\theta\cdot\log|\mathbb{F}|+\log w)\cdot\lambda)$ bits of local computation. This is summarised in an intermediate theorem, Theorem 19 below.

Theorem 19. *Assume the $\mathbb{F}$-$\mathsf{xLPN}(\theta,\theta'-\theta,\lambda/\theta')$ and $\mathbb{F}_2$-$\mathsf{LPN}(\lambda,O(\lambda),1/\lambda^{\delta})$ – where δ is any small enough constant – assumptions hold. Then there exists a semi-honest secure two-party protocol Π_{stp} which securely generates a subsets tensor powers correlation for subsets $(S_i)_{i\in[\mathsf{n_s}]}$ of $[w]$ and for which there exists a B-good cover comprised of m size-θ' masks, using the following resources:*

– Communication:
$$O\Big(\mathsf{n_s}\cdot\mathsf{poly}(\lambda)\cdot(1+B\lambda)^{\mathsf{tpp}}\cdot\log(1+B\theta')^{\mathsf{tpp}}\Big).$$
– Computation:
$$O\Big(\lambda\cdot\mathsf{n_s}\cdot(1+B\theta)^{\mathsf{tpp}}\cdot(1+B\theta')^{\mathsf{tpp}}\cdot(\log|\mathbb{F}|)^2 + \mathsf{n_s}\cdot\mathsf{poly}\big((1+B\lambda)^{\mathsf{tpp}}\cdot\log(1+B\theta')^{\mathsf{tpp}}\cdot\lambda\big) + m\cdot\theta'\cdot(\theta\cdot\log|\mathbb{F}|+\log w)\cdot\lambda\Big).$$

Wrapping up, using Π_{stp} with an appropriate good cover suffices to construct a protocol Π_{corr} for securely implementing the functionality $\mathscr{F}_{\mathsf{corr}}$. For each $i = 1\ldots\lceil d/k\rceil-1$, the parties need to generate a B-good cover of the $((U^{\mathsf{in}}_{i,j})_{j\in[\lceil s^{i+1}/\beta\rceil]},(U_{i,j})_{j\in[\lceil s^{i+1}/\beta\rceil]},(V^{\mathsf{in}}_{i,j})_{j\in[\lceil m^i/\beta\rceil]},(V_{i,j})_{j\in[\lceil m^i/\beta\rceil]})$ seen as subsets of $[n+s_i]$. A way to do so is to generate:

– a $B/2$-good cover A_{in} of $((U^{\mathsf{in}}_{i,j})_{j\in[\lceil s^{i+1}/\beta\rceil]},(V^{\mathsf{in}}_{i,j})_{j\in[\lceil m^i/\beta\rceil]})_{1\leq i<\lceil d/k\rceil}$ seen as subsets of $[n]$ comprised of M_{in} size-θ' masks;
– for each $i = 1\ldots\lceil d/k\rceil-1$, a $B/2$-good cover A_i of $((U_{i,j})_{j\in[\lceil s^{i+1}/\beta\rceil]},\ (V_{i,j})_{j\in[\lceil m^i/\beta\rceil]})$ seen as subsets of $[s_i]$ comprised of M_i size-θ' masks.

Let $\kappa,\kappa',\kappa_{\mathsf{in}}$ be correctness parameters. We set $M_{\mathsf{in}}\leftarrow\kappa_{\mathsf{in}}\cdot n\cdot\log n$, $M_i\leftarrow\kappa\cdot s_i\cdot\log s_i$ (which is upper-bounded by $M:=\kappa\cdot s\cdot\log s$), and $B\leftarrow 2\kappa'\cdot\kappa\cdot\ln s$. The probability $p=p(\kappa_{\mathsf{in}},\kappa,\kappa')$ all the above conditions are satisfied is then, by union-bound (and with $s_i/\beta\leq s/\beta$), at least:
$$1-\left(\frac{1}{n^{\kappa_{\mathsf{in}}-1}}+\frac{(s/k)/\beta}{n^{(\kappa'-2)\cdot\kappa_{\mathsf{in}}\cdot\theta/2}}\right)-\lceil d/k\rceil\cdot\left(\frac{1}{(s/k)^{\kappa-1}}+\frac{s/\beta}{(s/k)^{(\kappa'-2)\cdot\kappa\cdot\theta/2}}\right).$$
Wrapping-up, we get the following parametrised theorem.

Theorem 20. *Assume the $\mathbb{F}$-$\mathsf{xLPN}(\theta,\theta'-\theta,\lambda/\theta')$ and $\mathbb{F}_2$-$\mathsf{LPN}(\lambda,O(\lambda),1/\lambda^{\delta})$ – where δ is any small enough constant – assumptions hold. Then there exists a probabilistic semi-honest secure two-party protocol Π_{corr} which securely implements the functionality $\mathscr{F}_{\mathsf{corr}}$ given on Figure 3 with success probability:*
$$p^*=1-\left(\frac{1}{n^{\kappa_{\mathit{in}}-1}}+\frac{(s/k)/\beta}{n^{(\kappa'-2)\cdot\kappa_{\mathit{in}}\cdot\theta/2}}\right)-\lceil d/k\rceil\cdot\left(\frac{1}{(s/k)^{\kappa-1}}+\frac{s/\beta}{(s/k)^{(\kappa'-2)\cdot\kappa\cdot\theta/2}}\right).$$
Furthermore, it uses the following resources, where $B=2\kappa\kappa'\cdot\ln s$:

– Communication:

$$O\Big(\sum_{i=1}^{\lceil d/k\rceil-1}\frac{s_{i+1}+m_i}{\beta}\cdot\mathsf{poly}(\lambda)\cdot(1+B\lambda)^{2^k}\cdot\log(1+B\theta')^{2^k}\Big).$$

– Computation:

$$O\Bigg(\sum_{i=1}^{\lceil d/k\rceil-1}\frac{s_{i+1}+m_i}{\beta}\Big[\lambda\cdot(1+B\theta)^{\mathsf{tpp}}\cdot(1+B\theta')^{\mathsf{tpp}}\cdot(\log|\mathbb{F}|)^2$$
$$+\cdot\mathsf{poly}\big((1+B\lambda)^{\mathsf{tpp}}\cdot\log(1+B\theta')^{\mathsf{tpp}}\cdot\lambda\big)\Big]$$
$$+(\kappa_{in}\cdot n\log n\cdot\theta'\cdot(\theta\cdot\log|\mathbb{F}|+\log n)\cdot\lambda)$$
$$+(2\kappa\cdot s\log s\cdot\theta'\cdot(\theta\cdot\log|\mathbb{F}|+\log s)\cdot\lambda)\Bigg).$$

6 Choice of Parameters

In this section, we tune the parameters of our protocol. We want to ensure the scheme is correct with all but negligible probability, that it is secure, that the communication is sublinear, and that the computation is polynomial. We make two sets of choices for the parameters: the first optimising for communication (*i.e.* maximising the sublinearity factor), and the other for computation (and incidentally for the strength of the security assumption). Since the latter entails minimising the sublinearity factor k, we can can see any other choice as a tradeoff between the two, as a smooth continuum. Before we proceed, let us recall what the parameters are and how they are constrained.

6.1 Parameters.

$k=k(s)$ is the depth of the blocks in the *layer separation* phase; it is also the sublinearity factor of the final protocol. β is the number of outputs batched together for evaluation in a block, in the *output separation* phase. For the *input separation* phase, we need one B-good cover comprised of M_{in} size-θ' masks and $(\lceil d/k\rceil-1)$ B-good covers comprised of M size-θ' masks. For convenience we prefer to consider $\kappa_{\mathsf{in}}:=M_{\mathsf{in}}/(n\log n)$, $\kappa:=M/(n\log n)$, and $\kappa':=B/(2\kappa\ln s)$. λ is the security parameter in the $\mathbb{F}\text{-}\mathsf{xLPN}(\theta,\theta'-\theta,\lambda/\theta')$ assumption (which additionally introduces the parameter $\theta\le\theta'$) for the PCG and in the $\mathbb{F}_2\text{-}\mathsf{LPN}(\lambda,O(\lambda),1/\lambda^{\delta})$ – where δ is any small enough constant – assumption for the length-doubling PRG and the oblivious transfer.

6.2 Constraints.

1. *Correctness:* We have several choices on how to tune the parameters $\kappa,\kappa',\kappa_{\mathsf{in}}$:
   - $p>0$: If we choose the parameters so that p is non-zero, by a probabilistic argument, there must exist a choice of randomness which satisfies all the requirements. Then, if we so choose, we can preprocess circuits to find such a choice, or be satisfied with the non-constructive proof of existence of a sublinear protocol.
   - $p=\Omega(1)$: If we now choose them so that $1\ge p\ge c$ for some positive constant c, then the parties can sample fresh randomness until a combination of good covers is obtained, which occurs in an expected constant number of tries.
   - $p=1-o(1)$: Finally if p is taken to be close to 1, the protocol succeeds on the first try of sampling randomness with almost certain probability (or even with all but negligible probability if $p=1-\text{negl}$).

   We focus only on the last one, which is the most natural requirement. While it is the same

$$\left(\frac{1}{n^{\kappa_{\mathsf{in}}-1}}+\frac{(s/k)/\beta}{n^{(\kappa'-2)\cdot\kappa_{\mathsf{in}}\cdot\theta/2}}\right)+\lceil d/k\rceil\cdot\left(\frac{1}{(s/k)^{\kappa-1}}+\frac{s/\theta}{(s/k)^{(\kappa'-2)\cdot\kappa\cdot\theta/2}}\right)$$
$$=\frac{1}{s^{\omega(1)}}=2^{-\omega(\log s)}.$$

Since $\theta = \omega(1)$, κ, κ' are unconstrained (other than $\kappa' > 2$ and $\kappa > 1$) and can be taken to be constant. Therefore the constraint can be simplified to κ_{in} being subject to $\kappa_{\mathsf{in}} = \omega\left(\frac{\log s}{\log n}\right)$. Hence the following correctness constraints:

$$\begin{cases} M_{\mathtt{in}} = \omega(n \cdot \log s) \\ M = \Omega(s \cdot \log s) \\ B = \Omega(\log s) \end{cases} \tag{1}$$

2. *Security:* Security depends on two security assumptions $\mathbb{F}\text{-}\mathsf{xLPN}(\theta, \theta' - \theta, \lambda/\theta')$ (for the PCG) and $\mathbb{F}_2\text{-}\mathsf{LPN}(\lambda, O(\lambda), 1/\lambda^{\delta})$ – where δ is any small enough constant – (for the underlying PRG and OT). As we define the parameters θ, θ', λ, we will refine the exact flavour of assumption required.
3. *Communication:* The total communication of our protocol is the following:

$$O\Bigg(\underbrace{n + \frac{s}{k} + m}_{\substack{\text{Cost of}\\ \text{reconstructing}\\ \text{masked values } (n\\ \text{inputs, } \leq s/k\\ \text{intermediary gates,}\\ \text{and } m \text{ outputs)}}} + \overbrace{\frac{s/k+m}{\beta}}^{\substack{\text{Number of}\\ \text{batches of}\\ \beta \text{ nodes}\\ (i.e. \text{ number of}\\ \text{calls to PCG)}}} \times \underbrace{\mathsf{poly}(\lambda)}_{\substack{\text{Overhead}\\ \text{from}\\ \text{the PRG}\\ \text{(including}\\ \text{the OTs)}}} \times \overbrace{(1+B\lambda)^{2^k}}^{\substack{\text{Number of DPF}\\ \text{calls in MPFSS}}} \times \underbrace{\log\Big((1+B\theta')^{2^k}\Big)}_{\text{Seed for one DPF}} \Bigg).$$

We want this quantity to be $O(n + s/k + m)$, which translates to the following condition:

$$\beta = \Omega\left(\mathsf{poly}(\lambda) \cdot \Big(1 + B\lambda\Big)^{2^k} \cdot \log\left((1+B\theta')^{2^k}\right)\right). \tag{2}$$

4. *Computation:* The total computation is the following:

$$\begin{aligned} O\Bigg(\frac{s/k+m}{\beta} \Big[& \lambda \cdot (1+B\theta)^{\mathsf{tpp}} \cdot (1+B\theta')^{\mathsf{tpp}} \cdot \log|\mathbb{F}| \\ & + \cdot\mathsf{poly}\big((1+B\lambda)^{\mathsf{tpp}} \cdot \log(1+B\theta')^{\mathsf{tpp}} \cdot \lambda\big)\Big] \\ & + M_{\mathsf{in}} \cdot \theta' \cdot (\theta \cdot \log|\mathbb{F}| + \log n) \cdot \lambda + M \cdot \theta' \cdot (\theta \cdot \log|\mathbb{F}| + \log s) \cdot \lambda \Bigg). \end{aligned}$$

We want it to remain polynomial in s.

6.3 Choice of Parameters – Optimising for Communication.

Choosing k. Computation involves a term of the form B^{2^k} by eq. (2) with $B = \Omega(\log s)$ by eq. (1), which must remain polynomial in s.

$$(\log s)^{2^k} \leq s^{O(1)} \iff k \leq \log\log s - \log\log\log s + O(1).$$

Since k is the sublinearity factor, we also need it to be in $\omega(1)$, thus yielding the following constraint:

$$\omega(1) \leq k \leq \log\log s - \log\log\log s + O(1). \tag{3}$$

When we try and maximise k, it becomes more convenient to introduce the quantity $\epsilon := k/\log\log s$. Since we are only interested in asymptotic subinearity, we can strengthen the condition $\epsilon \leq 1 - \frac{\log\log\log s - O(1)}{\log\log s}$ to $\epsilon < \frac{1}{4}$ (this particular choice of constant will become apparent as we choose the other parameters as a function of k and hence of ϵ) without ultimately weakening the result. While it makes most sense in the vicinity of $O(\log\log s)$, it is well defined for the full spectrum of values of k:

$$\omega\left(\frac{1}{\log\log s}\right) \leq \epsilon < \frac{1}{4} \iff \omega(1) \leq k < \frac{\log\log s}{4}.$$

Choosing B, M_{in}, M. It makes sense so minimise these parameters, i.e. make them match the lower bounds of eq. (1). Therefore, (if we choose to set M_{in} to the explicit $n \cdot (\log s)^2$ rather than an arbitrary small $\omega(n \cdot \log s)$ for simplicity), with $\kappa \leftarrow 2$ and $\kappa' \leftarrow 3$:

$$\begin{cases} M_{\mathtt{in}} = n \cdot (\log s)^2 \\ M = 2 \cdot s \cdot \log s \\ B = 12 \cdot \log s \end{cases} \tag{4}$$

Choosing β. We choose to push β as high as it will go, *i.e.* we just need to ensure β^{2^k} remains polynomial in s (indeed in the general case, a batch of β nodes can have up to $2^k \cdot \beta$ ancestors in a depth-2^k circuit, hence we need $\beta \leq B\theta'$). This is the optimal choice if we want to prioritise communication complexity and security, but as we shall show, it will no hinder optimising for computation complexity.

$$\beta \leftarrow s^{O(1/2^k)} = 2^{O(\log^{1-\epsilon} s)}.$$

Choosing λ. Since λ determines the security of the LPN scheme it makes sense to maximise it, especially as it has no bearing on the asymptotic computation complexity. The only upper bound on λ is $\beta = \Omega((B\lambda)^{2^k+2} \cdot \log s)$ i.e. $(12 \cdot \log s \cdot \lambda)^{2^k+2} \cdot \log s = O(\beta) = O(s^{1/2^k})$. This translates to the following for λ:

$$\begin{aligned} \lambda &\leq \left(\tfrac{s^{O(1/2^{2k})}}{12 \cdot \log s}\right)^{O(1/(2^{2k}+2))} / \log s \\ &= 2^{O(\log^{1-2\epsilon} s - \log\log s \cdot \log^{\epsilon} s - \log\log s)} \\ &= 2^{O(\log^{1-2\epsilon} s)}, \text{ since } \epsilon < \tfrac{1}{4}. \end{aligned}$$

If we set $\lambda \leftarrow 2^{O(\log^{1-2\epsilon} s)}$ then $s = 2^{O(\log^{\frac{1}{1-2\epsilon}} \lambda)} = \lambda^{O(\log^{\epsilon'} \lambda)}$, where $\epsilon' := \frac{2\epsilon}{1-2\epsilon} \in]0,1[$. In other terms, if $\lambda \leftarrow (s/\log s)^{O(1/2^{2k})}$ then $s = O(2^{2k} \cdot (\lambda)^{O(2^{2k})} \cdot \log \lambda) = (\lambda)^{O(2^{2k})}$. Summarising:

$$\begin{cases} s = \lambda^{O(\log^{\epsilon'} \lambda)} \\ s = \lambda^{O(2^{2k})} \end{cases}$$

If we make sure $\theta' = \Theta(\theta)$, then this means that we can now choose the flavour of security assumption we need for $\mathbb{F}$-$\mathsf{xLPN}(\theta, \theta' - \theta, \lambda/\theta')$. If the adversary runs in time $\mathsf{poly}(s)$ then we want their advantage to be $1/s^{\omega(1)} = 1/2^{\mathsf{polylog}(\lambda)}$ which is the case if we assume the super-polynomial security of $\mathbb{F}$-$\mathsf{xLPN}(\theta, \theta', \lambda)$. Note however than if k (and therefore 2^{2k}) is taken to be an arbitrarily small $\omega(1)$ function, then we only need to assume its quasi-polynomial security (which smoothly transitions to polynomial as k tends to $\Theta(1)$).

Choosing θ and θ'. We assumed in the previous paragraph that $\theta' = \Theta(\theta)$, and we choose the constant θ'/θ so that $\mathbb{F}_2$-$\mathsf{LPN}(\lambda, O(\lambda), 1/\lambda^{\delta})$ – where δ is any small enough constant – is implied by $\mathbb{F}_2$-$\mathsf{xLPN}(\theta, \theta' - \theta, \lambda/\theta')$. The computation (and to a lesser extent the communication) grows in $B\theta'$ rather than in θ' alone, so the exact value of θ' isn't all that important (and can be compensated by B). Since $B\theta'$ must be at least $\beta 2^k$ in the general case (since all the ancestors of a batch of β outputs must be covered by the concatenation of B size-θ' masks) and it makes sense for B to be a positive integer, we want $\theta' \geq \beta 2^k$ We therefore set $\theta \leftarrow \beta \cdot 2^k = \log^{\epsilon} s \cdot 2^{O(\log^{1-\epsilon} s)}$.

Summary. We now summarise the choice of parameters which optimises for communication in fig. 8.

Parameter	Description	Choice for Optimal Comm.
k	Subl. Factor / Block Depth	$\epsilon \cdot (\log\log s),\ \epsilon \in]0, 1/4[$
β	Output batch size	$2^{O(\log^{1-\epsilon} s)}$
λ	Security Parameter	$2^{O(\log^{1-2\epsilon} s)}$
θ	–	$\beta \cdot \log^{\epsilon} s$
θ'	Mask Length in Good Cover	$O(\theta)$
κ_{in}	Defines M_{in}	$(\log s)^2 / \log n$
κ	Defines M	2
κ'	Defines B	3
B	"Goodness" of Cover	$12 \cdot \log s$
Total Communication:		$\Theta([n + s/\log\log s + m] \cdot \log \lvert\mathbb{F}\rvert)$
Total Computation:		$s^3 \cdot \mathsf{polylog}(s) \cdot (\log \lvert\mathbb{F}\rvert)^2$

Fig. 8: Choice of parameters for Main Theorem 1. The constants hidden in the exponents of β and λ are correlated, but omitted here for simplicity.

6.4 Choice of Parameters – Optimising for Computation and Security Assumption.

If we fix the sublinearity factor $k = \omega(1)$ to be some very slow-growing function (e.g. the inverse Ackermann function), we can set:

$$\beta \leftarrow s^{2^{-2k}};\ \lambda \leftarrow s^{2^{-3k}};\ \theta \leftarrow \beta \cdot 2^k;\ \theta' \leftarrow \mathsf{cst} \cdot \theta;\ \kappa \leftarrow 2;\ \kappa' \leftarrow 3;\ \kappa_{\mathsf{in}} \leftarrow \frac{(\log s)^2}{\log n}.$$

With this choice, we have $s = \lambda^{\omega(1)}$ where s need only be larger than any polynomial in λ, so the assumed security of LPN need only be *quasi-polynomial.* Using these parameters for our sublinear protocol, we get a quasi-linear amount of computation $s^{1+o(1)}$ (where the exact nature of the $o(1)$ function depends on how we tune the parameters), with a $(\log \lvert\mathbb{F}\rvert)^2$ overhead.

6.5 Wrapping-Up: Main Results with Optimal Choices of Parameters

Combining Theorems 12 and 20 (the former providing a secure protocol in the $\mathscr{F}_{\mathsf{corr}}$-hybrid model, and the latter instantiating the $\mathscr{F}_{\mathsf{corr}}$ functionality) with the parameter choice of Section 6.3 (and summarised in fig. 8) we get our main theorem, Main Theorem 1 below.

Main Theorem 1 (Sublinear Computation of Layered Circuits – Optimised for Communication)**.** *Assuming the super-polynomial security of*

- $\mathbb{F}$-LPN *with super-polynomial dimension ℓ, $O(\ell)$ samples, and inverse super-polynomial rate,*
- $\mathbb{F}_2$-LPN *with super-polynomial dimension $\ell' = s^{O((1))}$, $O(\ell')$ samples, and inverse polynomial rate (which is implied by the above if $\mathbb{F} = \mathbb{F}_2$),*

there exists a probabilistic semi-honest two-party protocol which securely evaluates any layered arithmetic circuit over $\mathbb{F}$ with success probability $1 - \mathrm{negl}(s)$ and which uses $O\left([n + s/\log\log s + m] \cdot \log \lvert\mathbb{F}\rvert\right)$ bits of communication and $s^3 \cdot \mathsf{polylog} s \cdot (\log \lvert\mathbb{F}\rvert)^2$ bits of computation (where s, n, and m are respectively the number of gates, inputs, and outputs of the circuit).

Instantiating the protocol with the parameters in Section 6.4 instead yields the following.

Main Theorem 2 (Sublinear Computation of Layered Circuits – Optimised for Computation)**.** *Assuming the quasi-polynomial security of*

- $\mathbb{F}$-LPN *with quasi-polynomial dimension ℓ, $O(\ell)$ samples, and inverse quasi-polynomial rate,*
- $\mathbb{F}_2$-LPN *with quasi-polynomial dimension ℓ', $O(\ell')$ samples, and inverse polynomial rate (which is implied by the above if $\mathbb{F} = \mathbb{F}_2$),*

there exists a probabilistic semi-honest two-party protocol which securely evaluates any layered arithmetic circuit over $\mathbb{F}$ with success probability $1 - \mathrm{negl}(s)$ and which uses $O\left([n + o(s) + m] \cdot \log \lvert\mathbb{F}\rvert\right)$ bits of communication and $s^{1+o(1)} \cdot (\log \lvert\mathbb{F}\rvert)^2$ bits of computation (where s, n, and m are respectively the number of gates, inputs, and outputs of the circuit).

References

AHI$^+$17. B. Applebaum, N. Haramaty, Y. Ishai, E. Kushilevitz, and V. Vaikuntanathan. Low-complexity cryptographic hash functions. pages 7:1–7:31, 2017.

AIK07. B. Applebaum, Y. Ishai, and E. Kushilevitz. Cryptography with constant input locality. pages 92–110, 2007.

AIK09. B. Applebaum, Y. Ishai, and E. Kushilevitz. Cryptography with constant input locality. 22(4):429–469, October 2009.

AJL$^+$12. G. Asharov, A. Jain, A. López-Alt, E. Tromer, V. Vaikuntanathan, and D. Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. pages 483–501, 2012.

BCG$^+$17. E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, and M. Orrù. Homomorphic secret sharing: Optimizations and applications. pages 2105–2122, 2017.

BCG$^+$19a. E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, P. Rindal, and P. Scholl. Efficient two-round OT extension and silent non-interactive secure computation. pages 291–308, 2019.

BCG$^+$19b. E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. pages 489–518, 2019.

BCG$^+$20a. E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl. Correlated pseudorandom functions from variable-density LPN. pages 1069–1080, 2020.

BCG$^+$20b. E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl. Efficient pseudorandom correlation generators from ring-LPN. pages 387–416, 2020.

BCGI18. E. Boyle, G. Couteau, N. Gilboa, and Y. Ishai. Compressing vector OLE. pages 896–912, 2018.

BFKL93. A. Blum, M. L. Furst, M. J. Kearns, and R. J. Lipton. Cryptographic primitives based on hard learning problems. In *Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings*, pages 278–291, 1993.

BFKL94. A. Blum, M. L. Furst, M. J. Kearns, and R. J. Lipton. Cryptographic primitives based on hard learning problems. pages 278–291, 1994.

BFKR91. D. Beaver, J. Feigenbaum, J. Kilian, and P. Rogaway. Security with low communication overhead. pages 62–76, 1991.

BGI15. E. Boyle, N. Gilboa, and Y. Ishai. Function secret sharing. pages 337–367, 2015.

BGI16a. E. Boyle, N. Gilboa, and Y. Ishai. Breaking the circuit size barrier for secure computation under DDH. pages 509–539, 2016.

BGI16b. E. Boyle, N. Gilboa, and Y. Ishai. Function secret sharing: Improvements and extensions. pages 1292–1303, 2016.

BI05. O. Barkol and Y. Ishai. Secure computation of constant-depth circuits with applications to database search problems. pages 395–411, 2005.

BKS19. E. Boyle, L. Kohl, and P. Scholl. Homomorphic secret sharing from lattices without FHE. pages 3–33, 2019.

BKW00. A. Blum, A. Kalai, and H. Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. pages 435–440, 2000.

BKW03. A. Blum, A. Kalai, and H. Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *Journal of the ACM (JACM)*, 50(4):506–519, 2003.

BLVW19. Z. Brakerski, V. Lyubashevsky, V. Vaikuntanathan, and D. Wichs. Worst-case hardness for LPN and cryptographic hashing via code smoothing. pages 619–635, 2019.

CG97. B. Chor and N. Gilboa. Computationally private information retrieval (extended abstract). pages 304–313, 1997.

CGKS95. B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. pages 41–50, 1995.

Cou19. G. Couteau. A note on the communication complexity of multiparty computation in the correlated randomness model. pages 473–503, 2019.

DFH12. I. Damgård, S. Faust, and C. Hazay. Secure two-party computation with low communication. pages 54–74, 2012.

DGH$^+$20. N. Döttling, S. Garg, M. Hajiabadi, D. Masny, and D. Wichs. Two-round oblivious transfer from cdh or lpn. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 768–797. Springer, 2020.

FGJS17. N. Fazio, R. Gennaro, T. Jafarikhah, and W. E. Skeith III. Homomorphic secret sharing from paillier encryption. pages 381–399, 2017.

Gen09. C. Gentry. Fully homomorphic encryption using ideal lattices. pages 169–178, 2009.

GI14. N. Gilboa and Y. Ishai. Distributed point functions and their applications. pages 640–658, 2014.

GMW87a. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. pages 218–229, 1987.

GMW87b. O. Goldreich, S. Micali, and A. Wigderson. How to prove all NP-statements in zero-knowledge, and a methodology of cryptographic protocol design. pages 171–185, 1987.

IKM$^+$13. Y. Ishai, E. Kushilevitz, S. Meldgaard, C. Orlandi, and A. Paskin-Cherniavsky. On the power of correlated randomness in secure computation. pages 600–620, 2013.

JKPT12. A. Jain, S. Krenn, K. Pietrzak, and A. Tentes. Commitments and efficient zero-knowledge proofs from learning parity with noise. pages 663–680, 2012.

KO97. E. Kushilevitz and R. Ostrovsky. Replication is NOT needed: SINGLE database, computationally-private information retrieval. pages 364–373, 1997.

Lyu05. V. Lyubashevsky. The parity problem in the presence of noise, decoding random linear codes, and the subset sum problem. In *Approximation, randomization and combinatorial optimization. Algorithms and techniques*, pages 378–389. Springer, 2005.

NN01. M. Naor and K. Nissim. Communication preserving protocols for secure function evaluation. pages 590–599, 2001.

OSY21. C. Orlandi, P. Scholl, and S. Yakoubov. The rise of paillier: Homomorphic secret sharing and public-key silent ot. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 678–708. Springer, 2021.

Pra62. E. Prange. The use of information sets in decoding cyclic codes. *IRE Transactions on Information Theory*, 8(5):5–9, 1962.

RS21. L. Roy and J. Singh. Large message homomorphic secret sharing from dcr and applications. 2021.

SGRR19. P. Schoppmann, A. Gascón, L. Reichert, and M. Raykova. Distributed vector-OLE: Improved constructions and implementation. pages 1055–1072, 2019.

Wak68. A. Waksman. A permutation network. *Journal of the ACM (JACM)*, 15(1):159–163, 1968.

Yao86. A. C.-C. Yao. How to generate and exchange secrets (extended abstract). pages 162–167, 1986.

YZW$^+$19. Y. Yu, J. Zhang, J. Weng, C. Guo, and X. Li. Collision resistant hashing from sub-exponential learning parity with noise. pages 3–24, 2019.