Multiparty Homomorphic Secret Sharing and More from LPN and MQ

Geoffroy Couteau[1], Naman Kumar[2] and Xiaxi Ye[3]

[1] Université Paris Cité, CNRS, IRIF
[2] Oregon State University
[3] Tsinghua University

Abstract. We give the first constructions of multiparty pseudorandom correlation generators, distributed point functions, and (negligible-error) homomorphic secret sharing for constant-degree polynomials for any number of parties without using LWE or iO. Our constructions are proven secure under the combination of LPN with dimension n, $2n$ samples, and noise rate $n^{\varepsilon-1}$ for a small constant ε, and MQ with n variables and $n^{1+\delta}$ equations.
As applications of our results, we obtain from the same assumptions secure multiparty computation protocols with sublinear communication and silent preprocessing, as well as private information retrieval for M servers and size-λ^d databases with optimal download rate and client-to-server communication $M^d \cdot \lambda^3$.

1 Introduction

Homomorphic Secret Sharing (HSS) is a primitive that allows two or more parties to perform a distributed computation on private inputs. Informally, an N-party HSS scheme randomly splits an input x into N shares on which the parties can then locally evaluate an arbitrary function f, with the outputs of the local computations corresponding to a secret sharing of $f(x)$, the computation on the private, shared input. First introduced in [BGI16a], HSS can be viewed as the secret sharing analogue of homomorphic encryption.

The work of [BGI16a] introduced a 2-party HSS scheme for the class of *Restricted Straight-line Multiplication (RMS) Programs* under the decisional Diffie-Hellman (DDH) assumption, which are powerful enough to encapsulate the general classes of $\mathbf{NC}^1$ circuits and polynomial-length branching programs. Since then, a long line of work has extended their results to form HSS schemes with improved efficiency [BGI17,BCG+17,ARS24], schemes from wider classes of assumptions [OSY21,RS21,ADOS22,BCG+19,CM21], and more recently to schemes that support more than two parties [COS+22,BCM23,DIJL23,CK24]. Various further works have shown the many applications of HSS to domains like sublinear secure computation, Private Information Retrieval (PIR), generating correlated randomness and more.

1.1 HSS for multiple parties

While various works have attempted to extend the techniques of [BGI16a] to multiple parties, until recently solutions to the general setting of $N \geq 3$ parties remained limited to constructions that used FHE or iO. This state of affairs was recently improved in two papers:

- In [DIJL23], the authors achieved an N-party HSS scheme that supports multivariate polynomials of degree $\log/\log\log$ for any N under a sparse variant of the LPN assumption. Unfortunately, a critical bottleneck inherent to their construction was the presence of *inverse-polynomial correctness error*, which makes it unsuitable for several applications, such as direct compilation into a sublinear MPC protocol (the authors of [DIJL23] do achieve a sublinear MPC protocol, however via a much less straightforward route) or constructing pseudorandom correlation generators.
- In [CK24], the authors developed a multiparty HSS scheme for the class of XOR-AND of constant-degree polynomials with no correctness error from a combination of the DDH/DCR and LPN assumptions along with an assumption on the existence of a constant-depth pseudorandom generator. However, this construction is limited to 8 parties.

In conclusion, non-FHE based techniques for multiparty HSS remain restricted to one of two constructions: a general N-party HSS from the sparse-LPN assumption that has the caveat of an inverse-polynomial correction error, or an HSS scheme for up to 8-parties using various combinations of assumptions along with LPN and constant-depth PRGs.

1.2 Sublinear secure computation in the multiparty setting

HSS has been used extensively in the design of protocols for *sublinear* secure computation, where the goal is to achieve per-party communication sublinear in the size of the circuit being computed. Traditional approaches to sublinear MPC relied on FHE. This state of affairs was changed in [BGI16a], who used HSS to obtain a 2-party secure computation scheme that supports arbitrary layered Boolean circuits with communication complexity of $O(s/\log s)$ bits. Since then, numerous developments have resulted in a rich landscape of constructions that imply sublinear MPC for arbitrary layered circuits. However the state of affairs for $N \geq 3$ parties is more limited.

- The work of [DIJL23] provides a construction of N-party sublinear MPC with $O(s/\log\log s)$ communication for any N from the sparse-LPN assumption.
- The work of [ARS24] presents a general protocol for sublinear MPC with $O(s/\log\log s)$ communication based on a primitive called *Succinct* HSS; this scheme does not yield an N-party HSS as an intermediary, however.
- A series of works based on the *nesting* approach [COS+22,BCM23,CK24] constructed sublinear MPC for an increasing number of parties and culminated in a 10-party protocol with $O(s/\log\log\log s)$ communication from one of DDH/DCR/LWE, an assumption on the existence of constant-depth pseudorandom generators and a superpolynomial variant of the LPN assumption.

1.3 Our contributions

In this work, we revisit the question of general HSS schemes for any N parties and improve upon this state of affairs. Our techniques follow upon and significantly improve the *nesting* strategy for HSS construction to yield improvements on many longstanding open problems. Our results form a culmination of a long line of (theoretical) works on low-communication MPC initiated by [BGI16a], which solves several important long-standing open problems (such as multiparty PIR without FHE or iO); expanding this technique to N-party HSS has been stated as a major open problem in multiple works.

An N-Party Homomorphic Secret Sharing Scheme from LPN and MQ. Our first contribution is the construction of an N-party HSS scheme that supports the class $\mathsf{XOR\text{-}AND}$ of constant-degree polynomials for arbitrary constant N based on the (standard) *Learning Parity with Noise* (LPN) and the *Multivariate Quadratic* (MQ) assumptions. Our scheme is the first HSS scheme for any arbitrary number of parties that supports negligible correctness error (beyond schemes based on Spooky encryption [DHRW16] or iO); we furthermore broaden the class of assumptions under which multiparty HSS can be constructed.

An N-Party Distributed Point Function Scheme from LPN and MQ. The key step in the construction of our N-HSS scheme is an iterative construction of N-party distributed point functions with short key size. Introduced in [BGI15], Distributed Point Functions (DPFs) form a significant subclass of general *Function Secret Sharing* schemes that allow parties to obtain small keys corresponding to some function f that allow distributed, local evaluation of their independent inputs to obtain output shares of the inputs evaluated on f. DPFs have wide applicability to several MPC protocols including PIR and the generation of correlated randomness. In this work, we improve upon known constructions of DPF to yield a new construction from the LPN and MQ assumptions for any constant N number of parties and with low key size.

New PCGs for Constant-Degree Correlations. As an immediate application of our N-party HSS for constant-degree polynomials, we obtain an N-party pseudorandom correlation generators for constant-degree correlations from LPN and MQ. In turn, this implies the existence of secure computation protocols with silent preprocessing for any constant-degree correlated randomness.

N-server PIR with Optimal Download Rate. Using our N-party distributed point function, we obtain from the same assumptions an N-party private information retrieval scheme for databases of size λ^d with client-to-server communication $O(N^d \cdot \lambda^3)$ and optimal download rate. To our knowledge, this is the first general multiparty PIR scheme that does not rely on advanced cryptographic primitives such as Spooky encryption of iO. Our scheme can be extended to other flavors of PIR, such as PIR by keyword, and PIR with payloads.

Sublinear MPC for N parties. Our final contribution is a new sublinear MPC protocol for N parties from superpolynomial variants of LPN and MQ, that significantly expands the number of assumptions from which sublinear MPC is known. Concretely, our scheme achieves a communication complexity of $O(s/\log^* s)$ per party, owing to complex parameter setting introduced by the recursive nesting approach. Our scheme is hence a direct improvement on the works of [CM21,CK24], directly reducing and streamlining the number of assumptions required to get sublinear MPC from the nesting approach to parallel the 2-party protocol of [CM21].

1.4 Organization

We provide an overview of our results in section 2. In section 4, we introduce the main building blocks on which our constructions are based. In section 5, we introduce our nesting strategy, and our boosting strategy. The outline of our work utilizes three kinds of results:

– Results imported from other work, such as Theorem 3. Here we import the theorem directly and provide a citation to the proof.
– Generalizations of previous theorems from 2 to N parties, as in Theorem 4 and Theorem 5. Here we provide a complete construction along with a formal proof of our generalization (the proof is a relatively simple generalization of the original 2-party proof).
– New constructions, as in Theorem 7 and Lemma 2, where we provide a full, formal treatment of the material.

Eventually, in section 6, we cover applications of our results: the first sublinear-communication M-server PIR without FHE or iO in Section 6.1, and applications to silent MPC and low-communication MPC in Section 6.2 and Section 6.3. Eventually, we sketch how our new results can be plugged in the framework of [CM21] to yield a sublinear-communication M-party secure computation protocol from LPN and MQ.

2 Technical Overview

Starting with the seminal work of [BGI16a], multiple homomorphic secret sharing schemes have been introduced [BGI17,BCG+17,BKS19,OSY21,RS21,ADOS22]. At a high level, all these schemes follow a common recipe (formalized in particular in the framework of [ADOS22]). Unfortunately, this approach to building HSS appears inherently stuck at the two-party barrier. This is due to the need for a *non-interactive share conversion* methodology, which can take different shapes (such as distributed discrete logarithm [BGI16a], conversion to shares over the integers [OSY21,RS21], rounding of shares [BKS19]) but works by design solely in the two-party setting (this was proven to be inherent in [BDIR18]). An exception to the above is the recent work of [DIJL23] which shows that, under the sparse LPN assumption, it is possible to build a multiparty HSS scheme. At a high level, this leverages the linearity of LPN to remove the need for share conversion in the first place, at the cost of introducing noise (the use of sparse matrices is crucial to control the noise growth). This is a strong result, but it comes with some downsides: it is only known from sparse LPN, and it only achieves HSS with imperfect correctness, which limits its applications (in particular, applications to pseudorandom correlation generators [BCG+19] require negligible correctness error).

An alternative line of work, initiated in [COS+22,BCM23] and recently refined in [CK24], seeks to overcome the two-party barrier via an approach called *nesting*. At a high level, the idea is the following: start from an "external" 2-party HSS scheme $(\mathsf{Share}^\star, \mathsf{Eval}^\star)$ for a function class $\mathcal{F}^\star$, and an "internal" 2-party

HSS scheme (Share, Eval) for a function class $\mathcal{F}$ such that for any function $f \in \mathcal{F}$, the function that evaluates f on a share x, denoted $g^\star : x \to \mathsf{Eval}(f, x)$, belongs to the class $\mathcal{F}^\star$ handled by the external scheme. Then one gets a 4-party HSS scheme for $\mathcal{F}$ by evaluating the internal HSS scheme inside the external HSS scheme. We sketch it below:

- Given an input x, share $(x_0, x_1) \leftarrow \mathsf{Share}(x)$ using the internal scheme.
- Share $(x_{b,0}, x_{b,1}) \leftarrow \mathsf{Share}^\star(x_b)$ using the external scheme. Each of the four parties receives one share.
- To evaluate f on a share $x_{b,b'}$, define $g^\star : x_b \to \mathsf{Eval}(f, x_b)$ and run $\mathsf{Eval}^\star(g^\star, x_{b,b'})$.

More generally, if one starts from an N-party external HSS and an M-party internal HSS, one gets an NM-party HSS. The main downside of this approach is that the complexity of evaluation blows up, as the final scheme must now homomorphically evaluate the evaluation algorithm of the internal scheme, and typically gets too complex to allow re-applying the nesting strategy. Hence, the first "nesting" constructions [COS+22,BCM23] were limited to 4 parties. Very recently, the work of [CK24] constructed an improved nesting strategy. At a high level, the approach of [CK24] starts from a simpler object—a 2-party distributed point function (DPF)—whose evaluation fits in a much lower complexity class. As a result, they managed to apply nesting *twice*, getting an 8-party DPF. Then, assuming LPN and using the compiler of [BCG+19,CM21], this DPF can be compiled back to an 8-party HSS. However, here again the complexity of the HSS scheme used in the nesting prevents pushing the limits further.

In light of this, it seems apparent that the nesting approach is inherently stuck at a small constant number of parties; indeed, it was the general opinion among recent work that a different route is needed to obtain N-party HSS. It is indeed not even apparent that there is anything to be found by working out a recursive nesting. However, we break through this limitation via a careful 'rescaling' approach, where we begin with a very carefully chosen parameter set (it is not even clear that this parameterization exists in the first place, although it is feasible to work out the details), then perform a careful, recursive rescaling that confines the exponential cost of nesting to the *constant in the exponent*. The precise rescaling technique is the main technical contribution of our work and unlocks a wide variety of applications.

In the remainder of our work, we introduce a new notion of 'programmability' for multiple parties, and show how our techniques can yield PCGs for any polynomial (instead of arbitrary constant) number of parties.

2.1 Our approach in a nutshell

We revisit the nesting strategy laid out in [CK24] and show how to unlock its full power to enable an *arbitrary constant number of nesting*. Our key insight is a powerful observation that is extremely simple in hindsight. Abstracting out, the technique in the work of [CK24] starts from a (low-complexity) two-party DPF and a (mild-complexity) 2-party HSS, which are combined to get a (mild-complexity) 4-party DPF. This result can then be combined again with the 2-party HSS (as mild-complexity is "mild enough" to still fit in the complexity class captured by the HSS) to get a high-complexity 8-party DPF, that is then converted (via LPN [BCG+19,CM21]) into an 8-party HSS.

We follow a similar strategy, but *get rid of the mild-complexity HSS* by interleaving the combinations with applications of the LPN-based DPF-to-HSS compilation. That is, our nesting strategy looks as follows:

1. Start from a low-complexity DPF (as guaranteed by [CK24]) and compile it directly to a *low-complexity* 2-party HSS via LPN. Note that the resulting HSS handles polynomials of a degree exponentially smaller than the complexity of the original DPF.
2. Combine this 2-party HSS with an "exponentially scaled-down" low-complexity 2-party DPF, getting a 4-party HSS for polynomials of degree "doubly-exponentially scaled down" compared to the original DPF.
3. Repeat the above process any constant number of times n to obtain a 2^n-party DPF. At the end, compile the 2^n-DPF back to a 2^n-HSS via LPN.

At this point, a reader familiar with [COS+22,BCM23,CK24] might wonder why this should work at all: indeed, the limiting factor in all previous constructions was a very similar requirement of "exponential downscaling": roughly, to handle degree-d polynomials, these works already required HSS for (low-depth) circuits of size λ^d, and adding one more layer would require handling super-exponentially-large circuits of size λ^{λ^d} ([CK24] added one more layer by ensuring that the degree-d polynomial was itself already computing a 2-party DPF in the first place). The core and surprising feature of our strategy is that it confines the exponential downscaling *to the constant in the exponent*. Concretely, starting from a 2-party DPF over a domain of size λ^{c^d}, we get a 4-party DPF over a domain of size λ^d, that is compiled back to a 4-party HSS for degree-d polynomials. Starting instead from a 2-party DPF over a domain of size $\lambda^{c^{c^d}}$ yields a 4-party DPF over a domain of size λ^{c^d}, hence an 8-party DPF over a domain of size λ^d, hence an 8-party HSS for degree-d polynomial, and so on. Given that the exponential loss is now confined to *the constant in the exponent*, one can therefore repeat this process an arbitrary (constant) number of times n, by starting from a DPF over a domain of size $\lambda^{(c,d)\uparrow n}$, where the notation $(c,d)\uparrow n$ denotes a size-n tower of exponents c with a d on top. We provide a representation of the general strategy in fig. 1. We let $\mathsf{DEG}(n)$ denote the class of functions computable by multivariate polynomials of total degree at most n.

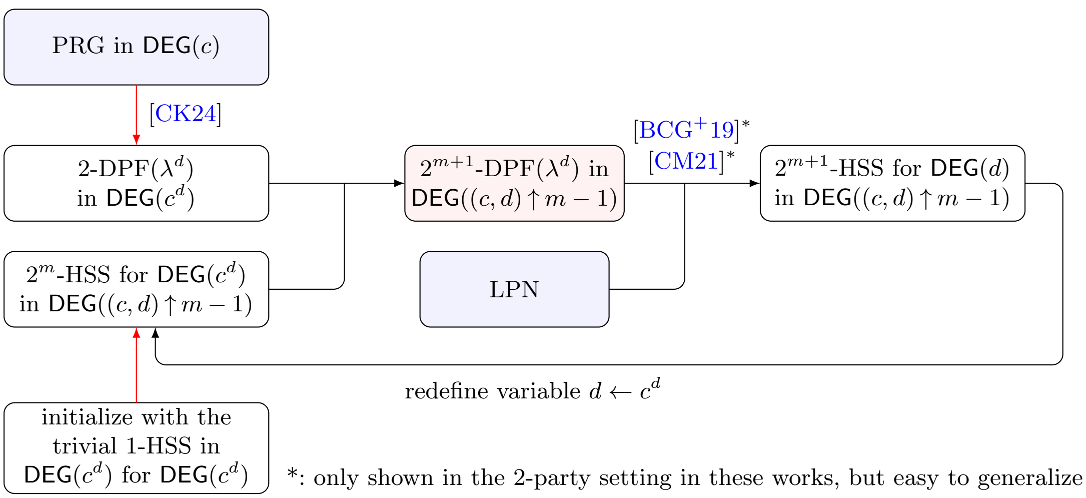

Fig. 1. Construction of a 2^n-party DPF in the class $\mathsf{DEG}(c^d)$ over the domain $[\lambda^d]$. The arrows in red denote the primitives used to initialize the recursive construction process. The blue boxes denote the cryptographic assumptions required for the construction, while the box in red indicates the target primitive. Due to the redefinition of d as c^d, each loop causes an exponential loss in the degree. Hence, to obtain a 2^n-DPF over the domain $[\lambda^d]$, the process must be initialized with a 2-DPF over the domain $[\lambda^{(c,d)\uparrow n}]$ computable in $\mathsf{DEG}(c^{(c,d)\uparrow n})$.

2.2 Details on the concrete parameters

Before moving on to the concrete nesting strategy, we briefly overview the assumptions and the primitives used in this work.

Assumptions. At a high level, we need to assume:

– The dual-$\mathbb{F}$-$\mathsf{LPN}(k, q, t)$ assumption states that given a random matrix $H \in \mathbb{F}^{(q-k)\times q}$ and a random vector $\boldsymbol{e} \in \mathbb{F}^q$ with at most t nonzero entries, no polynomial time adversary can distinguish $H \cdot \boldsymbol{e}$ from random. When $q = O(k)$ (as will be the case here), we note that the best known attacks on dual LPN run in time $2^{\Omega(t)}$. Dual LPN is equivalent to the (more common) primal form, that asserts that $A \cdot \boldsymbol{s} + \boldsymbol{e}$ cannot be distinguished from random when $A \leftarrow \mathbb{F}^{q\times k}$ and $\boldsymbol{s} \leftarrow \mathbb{F}^k$.

– The $\mathbb{F}$-$\mathsf{MQ}(n, m)$ assumption with n variables and m equations asserts that it is infeasible to solve (in polynomial time) a random system of m quadratic equations in n variables. In this work, we rely on a flavor of MQ with $m = n^{1+\varepsilon}$ for an arbitrary constant $\varepsilon > 0$. We note that no polynomial-time attack is known on MQ as long as $m \leq n^{2-\delta}$ for some constant $\delta > 0$.

In this work, we actually rely on the conjectured hardness of distinguishing an MQ instance from random (the decision version of the assumption), which is known to be polynomially reducible to the search version [BGP06]. We note that LPN is also known to be polynomially reducible to its search version in our parameter setting.

Primitives. Our nesting strategy alternates between three primitives:

– A 2^m-party Distributed Point Function (DPF) over a domain D is a pair $(\mathsf{Share}, \mathsf{Eval})$ such that for any point function $f_{\alpha,\beta}$ (that evaluates to 0 on all inputs, except for $f_{\alpha,\beta}(\alpha) = \beta$), $\mathsf{Share}(f_{\alpha,\beta})$ produces a 2^m-tuple of *short* keys $(k_i)_{i \leq 2^m}$ (whose description can grow polynomially with $|\alpha|, |\beta|$, but not with the size of the domain D). Informally, each strict subset of the keys hides α, β, yet the scheme allows computing shares of $f_{\alpha,\beta}$ on any input x: for all $x \in \mathsf{D}$, it holds that $\sum_i \mathsf{Eval}(k_i, x) = f_{\alpha,\beta}(x)$. Distributed point function are a special case of *Function Secret Sharing* (FSS), where the shared function f can come from a larger class.
– A 2^m-party Homomorphic Secret Sharing (HSS) for a function class $\mathcal{F}$ is a dual notion to FSS: Share allows sharing any input x into a 2^m-tuple $(x_1, \cdots, x_{2^m})$ such that for any $f \in \mathcal{F}$, $\sum_{i=1}^{2^m} \mathsf{Eval}(x_i, f) = f(x)$, yet any strict subset of shares hides x.
– Eventually, a 2^m-party Pseudorandom Correlation Generator (PCG) for an additive correlation C is a pair of algorithms $(\mathsf{Gen}, \mathsf{Expand})$ such that Gen produces a 2^m-tuple of short keys $(k_i)_{i \leq 2^m}$, and $\sum_i \mathsf{Expand}(k_i) = (\boldsymbol{r}, C(\boldsymbol{r}))$ for some vector $\boldsymbol{r}$ (that is, each k_i expands to a share of $\boldsymbol{r}$ and a share of $C(\boldsymbol{r})$). Security states that even given any strict subset S of the keys $(k_i)_{i \in S}$, the strings $(\mathsf{Expand}(k_i))_{i \notin S}$ must jointly look uniformly random conditioned on satisfying the correlation $\sum_i \mathsf{Expand}(k_i) = (\boldsymbol{r}, C(\boldsymbol{r}))$.

When C can be any n-variate degree-d polynomial, we say that the PCG is a PCG for n-variate degree-d polynomial correlations.

Parameter selection. We now overview the concrete parameters of the nesting strategy. Our main ingredient is a construction, introduced in [CK24], of a 2-party DPF over a domain of size λ^d (for an arbitrary constant d) with key size $\ell = d \cdot \lambda^2$ and evaluation procedure computable by an ℓ-variable multivariate polynomial of degree c^d. We call this a 2-DPF with parameters (λ^d, ℓ, c^d). The construction requires the existence of a PRG $G : \{0,1\}^\lambda \to \{0,1\}^{\lambda^2+\lambda}$ computable by a polynomial of degree c. In turn, such a PRG can be constructed assuming the multivariate quadratic (MQ) assumption with λ variables and $\lambda^{1+\varepsilon}$ equations (for any constant ε). For example, assuming MQ with $\lambda^{1.42}$ equations (a relatively conservative assumption) yields a construction with $c = 4$.

We first plug this 2-DPF in the compiler of [BCG+19]. Let λ denote the LPN dimension, 2λ be the code length, and $w = \lambda^{1/d}$ denote the noise weight. With these parameters, the best known attacks on LPN run in subexponential time $2^{\Omega(\lambda^{1/d})}$. Compiling the above 2-DPF with these LPN parameters yields a 2-party HSS for λ-variate degree-d polynomials, with shares of size $\lambda + d \cdot \lambda^3$.

Now, fix a new security parameter λ_0, a constant d_0, and consider the previous 2-DPF over a domain of size $\lambda_0^{d_0}$ with key size $\ell_0 = d_0 \cdot \lambda_0^2$ and evaluation procedure computable by an ℓ_0-variable multivariate polynomial of degree c^{d_0}. Define the following parameters:

$$d_1 = 3c^{d_0} \qquad \lambda_1 = \ell_0 \qquad w_1 = \lambda_0^{1/d_1}$$

and apply the compiler using $(\lambda_1, 2\lambda_1, w_1)$-LPN together with a 2-DPF with parameters $(\lambda_0^{d_1} > \lambda_1^{c^d}, \ell_1 = d_1 \cdot \lambda_0^2, d_2 = c^{d_1})$. This yields a 2-party HSS for ℓ_0-variate degree-c^{d_0} polynomials, with shares of size $\lambda_1 + \lambda_0 \cdot \ell_1 = O(\lambda_0^3)$. This scheme is powerful enough to allow homomorphic evaluation of the (degree-c^{d_0}) DPF evaluation procedure (on a key of length ℓ_0), which yields a 4-party DPF with key size $O(\lambda_0^3)$ over a domain of size $\lambda_0^{d_0}$. We continue the procedure by compiling back this 4-party DPF into a 4-party HSS, fix a 2-party DPF over $[\lambda_0^{d_0}]$ with key size ℓ_0 and degree d_0 evaluation, and rescale the parameters of the 4-party HSS appropriately to allow homomorphic evaluation of this DPF, obtaining an 8-party DPF. After m levels of this recursive strategy, carefully tracking down the evolution of the parameters after each rescaling, we obtain:

Theorem 1 (informal). *Fix integers $m, d > 0$. Assume the existence of a pseudorandom generator with polynomial stretch that can be computed by a degree-c polynomial and the dual $\mathbb{F}$-$\mathsf{LPN}(k, q = 2k, t)$ assumption with $k = O(\lambda^3)$ and $t = \lambda^{\varepsilon_{m,d}}$ where $\varepsilon_{m,c,d}$ is a constant depending on (m, c, d). Then, there is a 2^m-party distributed point function over the domain $[\lambda^d]$ with key size $O(\lambda^3)$ and whose evaluation algorithm can be computed on all inputs by a degree-$1/\varepsilon_{m,c,d}$ polynomial.*

In the above (informal) theorem, $\varepsilon_{m,c,d}$ is a tower of exponentials, roughly (but not exactly) a tower of m c's followed by a d on top.

2.3 Boosting to polynomially-many parties

The approach outlined in the previous section allows to construct 2^n-party DPFs and HSS for constant-degree polynomials, provided that n is a constant. We show how to boost the number of parties from constant to an arbitrary polynomial, by generalizing a technique introduced in [BCG+19] to construct multiparty pseudorandom correlation generators (PCGs) for degree-2 polynomials.

Let n be a constant and $N = N(\lambda)$ be a polynomial. Consider a 2^n-party DPF with key size ℓ over a domain of size λ^d. We apply the following sequence of transformations:

– Using $(\lambda, 2\lambda, \lambda^{1/d})$-LPN, compile the DPF into a 2^n-party HSS for λ-variate degree-d polynomials with shares of size $\lambda \cdot \ell$.
– Via [BCG+19], there is a two-way equivalence between (2^n-party) HSS and (2^n-party) PCG for constant-degree polynomials. Using the (simple) transformation from [BCG+19], we get a 2^n-party PCG for all degree-d correlations with seeds of length $\lambda \cdot \ell$.

From here, a core observation is that the PCG constructed this way satisfies *programmability*. Informally speaking (and restricting our attention to the two-party setting for simplicity), programmability says the following: consider three parties, Alice, Bob, and Carole, and let Alice and Bob receive PCG seeds a polynomial correlation: from their seeds, Alice obtains pseudorandom vectors $(\boldsymbol{r}_A, \boldsymbol{s}_A)$ and Bob obtains $(\boldsymbol{r}_B, \boldsymbol{s}_B)$, such that $\boldsymbol{s}_A + \boldsymbol{s}_B = P(\boldsymbol{r}_A + \boldsymbol{r}_B)$, where P is some fixed constant-degree multivariate polynomial. Programmability then says that $\boldsymbol{r}_A, \boldsymbol{r}_B$ have been computed deterministically (by Expand) from independent sections ρ_A, ρ_B of the random tape of Gen. That is, Gen takes a random tape of the form $R = (\rho_A, \rho_B, \rho)$ and defines $k_A = (f(\rho_A), g_A(R))$ and $k_B = (f(\rho_B), g_B(R))$ for some functions f, g_A, g_B. Then, Expand computes $\boldsymbol{r}_A = G_A(f(\rho_A))$, $\boldsymbol{r}_B = G_B(f(\rho_B))$ (where G_A, G_B are some specific PRGs). A consequence of this feature is that a party share can be *programmed* to remain the same across multiple pairs of parties. Concretely, running Gen on the random tape $R' = (\rho_A, \rho_C, \rho')$ yields $(k_A, k_C) = ((f(\rho_A), g_A(R')), (f(\rho_C), g_B(R')))$, implying that Alice obtains from Expand a pair $(\boldsymbol{r}_A, \boldsymbol{s}'_A)$ (and Bob obtains $(\boldsymbol{r}_C, \boldsymbol{s}_C)$, with $\boldsymbol{s}'_A + \boldsymbol{s}_C = P(\boldsymbol{r}_A + \boldsymbol{r}_C)$) where $\boldsymbol{r}_A$ is the *same* as Alice obtained in the instance with Bob.

Programmable PCGs are useful in that they allow correlating multiple 2-party PCG instances into a single *multiparty* instance. They were introduced (and constructed) in [BCG+19] in the two-party setting. In this work, we introduce and rely on a generalization of the notion for the multiparty setting, and we show that our nesting-based construction satisfies this generalized notion of programmability to the 2^m-party setting.

Then, fix a polynomial $M = M(\lambda)$ and let P denote an arbitrary M-variate degree-d polynomial. We will construct an N-party PCG for distributing additive shares of $(r, P(r))$ for a pseudorandom $r = (r_1, \cdots, r_M)$. Let $(r^{(1)}, \cdots, r^{(N)})$ denote the shares of r that the N parties will obtain. The main observation is the following: because P has degree d, $P(\sum_{i=1}^N r_i)$ can be written as a sum of $\binom{N}{d} \leq N^d$ polynomials P_k, where each polynomial is associated to a fixed size-d subset S_k of the parties, and operates only on the pseudorandom shares $(r_j^{(i)})_{i\in S_k}$ held by the parties in S_k. Then, we rely on a d-party programmable PCG to share eack P_k. The d-party PCG security ensures that each $r_j^{(i)}$ is pseudorandom, while programmability guarantees that the same pair (i,j) corresponds to the same share $r_j^{(i)}$ across all monomials, guaranteeing that the correct correlation is obtained. This immediately yields an N-party PCG for degree-d M-variate polynomial correlations, with keys of size $O(N^d \cdot \ell)$, where ℓ denotes the key size of the d-party PCG.

Combining this construction with the 2^m-party programmable PCG built on top of our 2^m-party HSS, we get an N-party pseudorandom correlation generators, that can be turned back into an N-party HSS via the two-way equivalence of [BCG+19] (this two-way equivalence was only shown in the two-party setting, but we prove in this work the straightforward extension to an arbitrary number of parties). Hence, we obtain:

Theorem 2 (informal). *Fix integers $m, d > 0$ and polynomials N, M. Assume the existence of a pseudorandom generator with polynomial stretch that can be computed by a degree-c polynomial and the dual $\mathbb{F}$-$\mathsf{LPN}(k, q = 2k, t)$ assumption with $k = O(\lambda^3)$ and $t = \lambda^{\varepsilon_{m,d}}$ where $\varepsilon_{m,c,d}$ is a constant depending on (m, c, d). Then, there is an N-party homomorphic secret sharing scheme for the class of degree-d M-variate polynomials, with shares of size $O(N^d \cdot \lambda^3 + M)$.*

3 Preliminaries

Notations. Given integers c, d, m, we let $(c, d) \uparrow m$ denote a tower of m c's followed by a d at the top. In particular, $(c, d) \uparrow 1 = c^d$, $(c, d) \uparrow 2 = c^{c^d}$ and so on. We define $(c, d) \uparrow 0 = d$. We also introduced a version where the exponents are scaled by a factor: given an integer b, we let $(b, c, d) \uparrow m$ be such that $(b, c, d) \uparrow 1 = b \cdot c^d$, $(b, c, d) \uparrow 2 = b \cdot c^{b \cdot c^d}$, $(b, c, d) \uparrow 3 = b \cdot c^{b \cdot c^{b \cdot c^d}}$, and so on. We say that a function $\mathsf{negl}\colon \mathbb{N} \to \mathbb{R}^+$ is *negligible* if it vanishes faster than every inverse polynomial.

We typically 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. Given $\boldsymbol{x}$ and $\boldsymbol{y}$ two vectors, we write $\boldsymbol{x} \otimes \boldsymbol{y}$ to denote the tensor product between $\boldsymbol{x}$ and $\boldsymbol{y}$, $\boldsymbol{x}^{\otimes 2}$ for $\boldsymbol{x} \otimes \boldsymbol{x}$, and more generally, $\boldsymbol{x}^{\otimes n}$ for the *n-th tensor power* of $\boldsymbol{x}$, $\boldsymbol{x} \otimes \boldsymbol{x} \otimes \cdots \otimes \boldsymbol{x}$. We let $\boldsymbol{x}||\boldsymbol{y}$ denote the (column) vector obtained by their concatenation. Given a vector $\boldsymbol{x}$ of length $|\boldsymbol{x}| = n$, the notation $\mathsf{HW}(x)$ denotes the Hamming weight $\boldsymbol{x}$.

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 labeled according to variables $\{x_1, \cdots, x_n\}$; the *(computation) gates* are labeled 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*. An arithmetic circuit is *layered* if its gates 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 $i \leq d - 1$.

3.1 Learning Parity with Noise

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

$$\begin{aligned}&\{(A, \mathbf{b}) \mid A \leftarrow\$\ \mathbb{F}^{q\times k}, \mathbf{e} \leftarrow\$\ \mathsf{Ber}_r(\mathbb{F})^q, \mathbf{s} \leftarrow\$\ \mathbb{F}^k, \mathbf{b} \leftarrow A\cdot\mathbf{s} + \mathbf{e}\}\\ \approx_c &\{(A, \mathbf{b}) \mid A \leftarrow\$\ \mathbb{F}^{q\times k}, \mathbf{b} \leftarrow\$\ \mathbb{F}^q\}.\end{aligned}$$

Our construction will mostly rely on a variant of LPN, called *exact LPN* (denoted $\times\mathsf{LPN}$) [JKPT12]. In this variant, the noise vector $\mathbf{e}$ 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 *exact* rq nonzero coordinates (while a sample from $\mathsf{Ber}_r(\mathbb{F})^q$ has an *expected* number rq of nonzero coordinates). In the following, we will still use $\mathbb{F}$-$\mathsf{LPN}(k, q, rq)$ to denote the $\mathbb{F}$-$\times\mathsf{LPN}(k, q, rq)$ assumption. We will also base our construction on the following dual LPN assumption.

Definition 2 (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-$\mathbb{F}$-$\mathsf{LPN}(k, q, rq)$ assumption states that*

$$\{(H, \mathbf{b}) \mid H \leftarrow\$ \mathbb{F}^{(q-k)\times q}, \mathbf{e} \leftarrow\$ \mathsf{Ber}_r(\mathbb{F})^q, \mathbf{b} \leftarrow H \cdot \mathbf{e}\}$$
$$\approx_c \{(H, \mathbf{b}) \mid H \leftarrow\$ \mathbb{F}^{(q-k)\times q}, \mathbf{b} \leftarrow\$ \mathbb{F}^q\}.$$

It is clear that solving dual LPN assumption is at least as hard as solving LPN.

3.2 Multivariate Quadratic Assumption

We represent a multivariate polynomial with n variables over a finite field $\mathbb{F}_q$ as $Q[\mathbf{x}]$, where

$$Q[\mathbf{x}] = \sum_{1\le j\le k\le n} \alpha_{j,k} x_j x_k + \sum_{1\le j\le n} \beta_j x_j + \gamma.$$

Here $\mathbf{x} = (x_1, \ldots, x_n)$ is a vector of n variables, and $\alpha_{j,k}, \beta_j, \gamma \in \mathbb{F}_q$ are the coefficients to the corresponding monomials. A multivariate quadratic system is a set of multivariate quadratic polynomials Let $R = (R_i)_{i\in[m]} \in (\mathbb{F}_q^{n\times n})^m$ be m $n \times n$ matrices, $L \in \mathbb{F}_q^{m\times n}$ be an $m \times n$ matrix, and $\mathbf{d} \in \mathbb{F}_q^m$ be a vector. We can write each polynomial as $Q_i[\mathbf{x}] = \mathbf{x}^\top \cdot R_i \cdot \mathbf{x} + L_i \cdot \mathbf{x} + d_i$. We denote the system by $S[\mathbf{x}] = R[\mathbf{x}] + L[\mathbf{x}] + \mathbf{d}$.

Definition 3 (Multivariate Quadratic Assumption). *Let n, m, q be parameters such that q is a prime, χ is a distribution on $(\mathbb{F}_q^{n\times n})^m$ and let $H \subseteq \mathbb{F}_q$. We denote by $MQ(n, m, q, \chi, H)$ on an instance $(S, S[\mathbf{x}])$ to be the multivariate quadratic problem, such that the goal of a solver is to output some $\mathbf{x}' \in \mathbb{F}_q^n$ such that $S[\mathbf{x}'] = S[\mathbf{x}]$, where $S = (R, L, \mathbf{d})$ with $R \leftarrow \chi, L \leftarrow \mathbb{F}_q^{m\times n}, \mathbf{d} \leftarrow \mathbb{F}_q^m$ and $\mathbf{x} \leftarrow H^n$.*

Let λ be the security parameter. For every constant $c > 1 \in \mathbb{N}$, every efficiently computable and polynomially bounded $n, m, q : \mathbb{N} \to \mathbb{N}, \alpha : \mathbb{N} \to [-q/2, q/2]$ and every $0 < \beta \le [q/2]$ such that $m = cn$, $\alpha = O(1)$, let Φ_α be the distribution over $(\mathbb{F}_q^{n\times n})^m$ with each element sampled identically and independently from discrete Gaussian distributions D_α's with mean 0, standard deviation α, namely each D_α samples $z \pmod q \leftarrow N(0, \alpha^2)$, and let $H_\beta = [-\beta, \beta]$. Then for every PPT solver $\mathcal{A}$, there exists some negligible function $\mathsf{negl}(\cdot)$ such that the following holds for all sufficiently large λ:

$$\Pr_{S\leftarrow MQ(n,m,q,\Phi_\alpha,H_\beta),\mathbf{x}\leftarrow H_\beta^n}[\mathbf{x}' \leftarrow \mathcal{A}(S, S[\mathbf{x}]) : S[\mathbf{x}'] = S[\mathbf{x}]] < \mathsf{negl}(\lambda).$$

3.3 Function Secret Sharing

A point function with input domain $[D]$ and outputs in a group $\mathbb{G}$ is a function $f_{\alpha,\beta} : [D] \mapsto \mathbb{G}$ such that $f_{\alpha,\beta}(x) = \beta$ if $x = \alpha$, and 0 otherwise. Informally, a DPF is a pair of algorithms $(\mathsf{Gen}, \mathsf{Eval})$ which shares a point function f into N shares $(K_1, \cdots, K_N) \leftarrow \mathsf{DPF.Gen}(1^\lambda, f)$ such that (correctness) on any input x, the values $(y_1, \cdots, y_N)$ defined as $y_i \leftarrow \mathsf{DPF.Eval}(i, K_i, x)$ form additive shares of $f(x)$, and (security) for any subset $T \subsetneq [N]$, the set of keys $(K_\sigma)_{\sigma\in T}$ computationally hides (α, β).

Definition 4 (Distributed point functions [GI14,BGI16b]). *An N-party* distributed point function *(DPF) scheme with input domain $[D]$ and output domain an abelian group $(\mathbb{G}, +)$, is a pair of PPT algorithms $\mathsf{DPF} = (\mathsf{DPF.Gen}, \mathsf{DPF.Eval})$ with the following syntax:*

– $\mathsf{DPF.Gen}(1^\lambda, \alpha, \beta)$, *given security parameter λ and description of a point function $f_{\alpha,\beta}$, outputs N keys $(K_1, \cdots, K_N)$;*
– $\mathsf{DPF.Eval}(i, K_i, x)$, *given party index $i \in [N]$, key K_i, and input $x \in [D]$, outputs a group element $y_i \in \mathbb{G}$.*

The scheme DPF *should satisfy the following requirements:*

- ***Correctness:*** *For any* $(\alpha, \beta) \in [D] \times \mathbb{G}$ *and* $x \in [D]$*, we have*

$$\Pr[(K_1, \cdots, K_N) \leftarrow\$ \ \mathsf{DPF.Gen}(1^\lambda, f) : \sum_{i \in [N]} \mathsf{DPF.Eval}(i, K_i, x) = f(x)] = 1.$$

- ***Security:*** *For every set of corrupted parties* $T \subsetneq [N]$*, there exists a PPT simulator* Sim *such that for any family* $f_{\alpha,\beta} = \{f_{\alpha_\lambda,\beta_\lambda} : [D(\lambda)] \to \mathbb{G}_\lambda\}_{\lambda \in \mathbb{N}}$ *of point functions over domain* $D(\lambda)$ *and group* $\mathbb{G}_\lambda$*, the distributions*

$$\{(K_j)_{j \in T} \mid (K_1, \cdots, K_N) \leftarrow\$ \ \mathsf{DPF.Gen}(1^\lambda, \alpha_\lambda, \beta_\lambda)\}$$

and

$$\{(K_j)_{j \in T} \mid (K_j)_{j \in T} \leftarrow\$ \ \mathsf{Sim}(1^\lambda, D(\lambda), \mathbb{G}_\lambda)\}$$

are computationally indistinguishable.

Given a DPF scheme ($\mathsf{DPF.Gen}, \mathsf{DPF.Eval}$)*, we denote by* $\mathsf{DPF.FullEval}$ *an algorithm which, on input a party index* i*, and an evaluation key* K_i*, outputs the* D*-tuple* $(\mathsf{DPF.Eval}(i, K_i, j))_{j \leq D} \in \mathbb{G}^D$*. Eventually, we say that a distributed point function* DPF *is* weakly efficient *if the running time of* $\mathsf{DPF.Gen}$ *is allowed to depend polynomially on the domain size* D*.*

In this work, we will also rely on FSS for *multi-point functions* (MPFSS), which are essentially sums of point functions. An MPFSS scheme for t-point functions can be obtain easily from t instances of a DPF.

3.4 Homomorphic Secret Sharing

We consider a definition of Homomorphic Secret Sharing with simulation-based security guarantee, with "leakage" corresponding to the input length n.

Definition 5 (Homomorphic Secret Sharing [BGI16a]). *An* N*-party* Homomorphic Secret-Sharing (HSS) *scheme (with additive reconstruction) for a class* $\mathcal{F}$ *of functions over a finite field* $\mathbb{F}$ *is a pair of algorithms* $\mathsf{HSS} = (\mathsf{HSS.Share}, \mathsf{HSS.Eval})$ *with the following syntax and properties:*

- $\mathsf{Share}(1^\lambda, x)$*: On input* 1^λ *(the security parameter) and* $x \in \mathbb{F}^{n(\lambda)}$ *(the input), the* sharing algorithm Share *outputs* N *input shares* $(x^{(1)}, \ldots, x^{(N)})$*.*
- $\mathsf{Eval}(i, f, x^{(i)})$*: On input* $i \in [N]$ *(the party index),* $f \in \mathcal{F}$ *(the function to be homomorphically evaluated, implicitly assumed to specify input and output lengths* n, m*), and* $x^{(i)}$ *(the* i*-th input share), the* evaluation algorithm Eval *outputs the* i*-th output share* $y^{(i)} \in \mathbb{F}^m$*.*

- ***Correctness:*** *For any* 1^λ*, input* $x \in \mathbb{F}^{n(\lambda)}$*, and any function* $f \in \mathcal{F}$*,*

$$\Pr\left[y^{(1)} + \cdots + y^{(N)} = f(x) : \begin{array}{l} (x^{(1)}, \ldots, x^{(N)}) \leftarrow\$ \ \mathsf{HSS.Share}(1^\lambda, x) \\ y^{(i)} \leftarrow\$ \ \mathsf{HSS.Eval}(i, f, x^{(i)}), \ i = 1 \ldots N \end{array} \right] = 1 \ .$$

- ***Security:*** *For every set of corrupted parties* $T \subsetneq [N]$*, there exists a PPT algorithm* Sim*, such that for every sequence of inputs* $x_1, x_2, \ldots \in \mathbb{F}^{n(\lambda)}$ *the outputs of the following experiments* Real *and* Ideal *are computationally indistinguishable:*
  - $\mathsf{Real}(1^\lambda)$*: run* $(x^{(1)}, \ldots, x^{(N)}) \leftarrow\$ \ \mathsf{HSS.Share}(1^\lambda, x_\lambda)$ *and output* $(x^{(\sigma)})_{\sigma \in T}$*.*
  - $\mathsf{Sim}(1^\lambda)$*: output* $\mathsf{Sim}(1^\lambda, 1^N, 1^n)$*.*

We say that a homomorphic secret sharing scheme is compact *if there exists a fixed polynomial* p *such that for every input* x*, the size (in bits) of each share* $x^{(i)}$ *of* x *is at most* $O(|x|) + p(\lambda)$*.*

Remark 1 (Compact Single-function HSS). A single-function HSS is an HSS scheme for a singleton function class. Let $\mathcal{F}$ be a (not necessarily singleton) function class. We say there exists *compact single-function* HSS for any function in $\mathcal{C}$, if for every $f : \mathbb{F}^n \to \mathbb{F}^m \in \mathcal{F}$ there exists an HSS scheme HSS_f for $\{f\}$ such that the circuit-size of $\mathsf{HSS}_f.\mathsf{Share}$ is a fixed polynomial in n (and otherwise independent of f).

This notion can be seen as a weakening of compact HSS for $\mathcal{C}$ where the function to be homomorphically evaluated is known when running the sharing algorithm.

3.5 Pseudorandom Correlation Generators

Introduced in [BCG+19], pseudorandom correlation generators (PCG) allow to generate a short key for each party P_σ which can be expanded to a long string R_σ. The correctness of PCG states that the expanded long strings correctly satisfy the target correlation. The security of PCG requires that for any set $T \subsetneq [N]$ of corrupted parties, given the keys $(k_\sigma)_{\sigma\in T}$, the distribution of expanded strings $(R_{\tilde\sigma})_{\tilde\sigma\in[N]\setminus T}$ using honest values $(k_{\tilde\sigma})_{\tilde\sigma\in[N]\setminus T}$ is indistinguishable from that of randomly sampled strings conditioned on satisfying the target correlation. To define the security, we need to characterize the so-called *reverse sampleable* property. The definitions are given as below.

Definition 6 (Correlation Generator). *A PPT algorithm $\mathcal{C}$ is called a correlation generator for N parties, if $\mathcal{C}$ on input 1^λ outputs N elements in $(\{0,1\}^n)^N$ for $n \in \mathsf{poly}(\lambda)$.*

Definition 7 (Reverse-sampleable Correlation Generator). *Let $\mathcal{C}$ be an N-party correlation generator. We say $\mathcal{C}$ is reverse sampleable if there exists a PPT algorithm* RSample *such that for $T \subsetneq [N]$ the correlation obtained via*

$$\left\{ (R'_1, R'_2, \ldots, R'_N) \;\middle|\; \begin{array}{r} (R_1, R_2, \ldots, R_N) \leftarrow\$\ \mathcal{C}(1^\lambda), \\ R'_\sigma := R_\sigma \textit{ for } \sigma \in T, \\ (R'_{\tilde\sigma})_{\tilde\sigma\in[N]\setminus T} \leftarrow\$\ \mathsf{RSample}(T, (R_\sigma)_{\sigma\in T}) \end{array} \right\}$$

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

Definition 8 (Pseudorandom Correlation Generator(PCG)). *Let $\mathcal{C}$ be a reverse-sampleable correlation generator. A pseudorandom correlation generator (PCG) for $\mathcal{C}$ is a pair of algorithms* (PCG.Gen, PCG.Expand) *with the following syntax:*

- PCG.Gen(1^λ) *is a PPT algorithm that given a security parameter λ, outputs N seeds $(k_1, k_2, \ldots, k_N)$;*
- PCG.Expand(σ, k_σ) *is polynomial-time algorithm that given party index $\sigma \in [N]$ 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:*

  $$\left\{ (R_1, R_2, \ldots, R_N) \;\middle|\; \begin{array}{r} (k_1, k_2, \ldots, k_N) \leftarrow \mathsf{PCG.Gen}(1^\lambda) \\ R_\sigma \leftarrow \mathsf{PCG.Expand}(\sigma, k_\sigma) \textit{ for } \sigma \in [N] \end{array} \right\}$$

  is computationally indistinguishable from $\mathcal{C}(1^\lambda)$.
- ***Security.*** *For any $T \subsetneq [N]$, the following two distributions*

  $$\left\{ (k_\sigma)_{\sigma\in T}, (R_{\tilde\sigma})_{\tilde\sigma\in[N]\setminus T} \;\middle|\; \begin{array}{r} (k_1, k_2, \ldots, k_N) \leftarrow \mathsf{PCG.Gen}(1^\lambda) \\ R_{\tilde\sigma} \leftarrow \mathsf{PCG.Expand}(\tilde\sigma, k_{\tilde\sigma}) \textit{ for } \tilde\sigma \in [N] \setminus T \end{array} \right\}$$

  and

  $$\left\{ (k_\sigma)_{\sigma\in T}, (R_{\tilde\sigma})_{\tilde\sigma\in[N]\setminus T} \;\middle|\; \begin{array}{r} (k_1, k_2, \ldots, k_N) \leftarrow \mathsf{PCG.Gen}(1^\lambda) \\ R_\sigma \leftarrow \mathsf{PCG.Expand}(\sigma, k_\sigma) \textit{ for } \sigma \in T \\ (R_{\tilde\sigma})_{\tilde\sigma\in[N]\setminus T} \leftarrow \mathsf{RSample}(T, (R_\sigma)_{\sigma\in T}) \end{array} \right\}$$

  are computationally indistinguishable, where RSample *is the reverse sampling algorithm for correlation $\mathcal{C}$.*

4 Conversions Between DPF, HSS, and PCG

In this section, we overview protocols for converting between DPF, PCG, and HSS. At a high level, the DPF-to-PCG construction relies on the LPN assumption and yields a PCG for constant-degree polynomials correlations; PCGs for constant-degree polynomials, in turn, can be shown to be equivalent to HSS for constant-degree polynomials. Then, combining an M-party HSS for constant-degree polynomials with an N-party DPF computable by a constant-degree polynomial yields an NM-party DPF.

All conversions discussed in this section have been introduced in previous works [BCG+19,CK24], albeit sometimes only in the setting of two parties. We reproduce the constructions for completeness, generalize them to the multiparty setting, and (when the original proof was restricted to two parties) prove their security.

4.1 Low Complexity 2-party DPF

We start from a low-complexity 2-party DPF constructed in [CK24] presented in Protocol 1, which is built based on the existence of PRGs computable by low degree polynomials. The construction of distributing a point function $f_{\alpha,\beta} : [\lambda^d] \to \mathbb{F}_p$ of domain size λ^d is based on GGM tree structure following [BGI15,BGI16b]. As observed in [CK24], the resulting computation complexity of the evaluation algorithm for DPF is determined by the following two parameters,

- the depth of the evaluation tree and
- the degree of the using PRG.

Considering the first parameter, [CK24] makes use of a λ-ary GGM tree structure instantiated by a PRG stretching λ bits into $\lambda^2 + \lambda$ bits rather than the original 2-ary GGM tree structure in [BGI15,BGI16b], which leads to an evaluation tree of depth $\log_\lambda \lambda^d = d$. As for the second parameter, we require a PRG with polynomial stretch is computable by a polynomial of constant degree c, which exists assuming the family of Multivariate Quadratic (MQ) assumptions.

As a result, the constructed DPF for point functions of domain size λ^d has key length of $O(d \cdot \lambda^2)$ bits and can be evaluated by a polynomial of constant degree c^d, which will be referred as a 2-DPF with parameters (λ^d, ℓ, c^d) with ℓ denoting the key size. We present the construction in Protocol 1 whose functionality and the computation complexity are concluded in Theorem 3 (Theorem 11 in [CK24]).

Theorem 3. *Suppose that $G : \{0,1\}^\lambda \to \{0,1\}^{\lambda\cdot(\lambda+1)}$ is a pseudorandom generator with polynomial stretch which can be computed by a degree-c polynomial. Then the scheme* $(\mathsf{DPF.Gen}, \mathsf{DPF.Eval})$ *from Protocol 1 is a 2-party DPF for the family of point functions $\{f_{\alpha,\beta} : [\lambda^d] \to \mathbb{F}_p \mid \alpha \in [\lambda^d], \beta \in \mathbb{F}_p\}$ with the following properties.*

- DPF.Gen *outputs N keys, each of $d \cdot \lambda \cdot (\lambda + 1) + \lambda + \lceil \log_2 |\mathbb{F}_p| \rceil$ bits, and involves at most $2d$ invocations of G and $O(d\lambda^2)$ additional Boolean operations.*
- DPF.Eval *can be computed by polynomials of degree c^d and involves at most d invocations of G.*

Protocol 1: Compact Distributed Point Function

Parameters: Let $G : \{0,1\}^\lambda \to \{0,1\}^{\lambda(\lambda+1)}$ be pseudorandom generator computable by a degree-c polynomial. Let $\mathsf{Convert} : \{0,1\}^\lambda \to \mathbb{F}_p$ be a map converting any λ-bit string to a pseudorandom field element of $\mathbb{F}_p$.

$\mathsf{DPF.Gen}(1^\lambda, \alpha, \beta, \mathbb{F}_p)$:

1. Let $\alpha = (\alpha_1, \ldots, \alpha_d) \in \{0,1\}^{d\cdot\log\lambda}$ be the decomposition of α into d substrings of $\log\lambda$-bit length.
2. Sample random $s_0^{(0)} \leftarrow \{0,1\}^\lambda$ and $s_1^{(0)} \leftarrow \{0,1\}^\lambda$.
3. Let $t_0^{(0)} = 0$ and $t_1^{(0)} = 1$.
4. For $i = 1$ to d do
   (a) $s_0^1||t_0^1||\ldots||s_0^\lambda||t_0^\lambda \leftarrow G(s_0^{(i-1)}), s_1^1||t_1^1||\ldots||s_1^\lambda||t_1^\lambda \leftarrow G(s_1^{(i-1)})$.

(b) Set $\mathsf{Keep} \leftarrow \alpha_i$.
(c) Set $s_{\mathsf{CW}}^{\mathsf{Keep}} \leftarrow_{\$} \{0,1\}^{\lambda}$ and $s_{\mathsf{CW}}^{j} \leftarrow s_0^j \oplus s_1^j$ for other $j \in [\lambda]$.
(d) Set $t_{\mathsf{CW}}^{\mathsf{Keep}} \leftarrow t_0^{\mathsf{Keep}} \oplus t_1^{\mathsf{Keep}} \oplus 1$ and $t_{\mathsf{CW}}^{j} \leftarrow t_0^j \oplus t_1^j$ for other $j \in [\lambda]$.
(e) $\mathsf{CW}^{(i)} \leftarrow s_{\mathsf{CW}}^1 || t_{\mathsf{CW}}^1 || \ldots || s_{\mathsf{CW}}^{\lambda} || t_{\mathsf{CW}}^{\lambda}$.
(f) $s_{\sigma}^{(i)} = s_{\sigma}^{\mathsf{Keep}} \oplus t_{\sigma}^{(i-1)} \cdot s_{\mathsf{CW}}^{\mathsf{Keep}}$ for $\sigma \in \{0,1\}$.
(g) $t_{\sigma}^{(i)} = t_{\sigma}^{\mathsf{Keep}} \oplus t_{\sigma}^{(i-1)} \cdot t_{\mathsf{CW}}^{\mathsf{Keep}}$ for $\sigma \in \{0,1\}$.

5. $\mathsf{CW}^{(d+1)} \leftarrow \beta - \mathsf{Convert}(s_0^{(d)}) - \mathsf{Convert}(s_1^{(d)}) \in \mathbb{F}_p$.
6. Let $k_{\sigma} = s_{\sigma}^{(0)} || \mathsf{CW}^{(1)} || \ldots || \mathsf{CW}^{(d+1)}$ for $\sigma \in \{0,1\}$.
7. Return (k_0, k_1).

$\mathsf{DPF.Eval}(b, k_{\sigma}, x)$:

1. Let $x = (x_1, \ldots, x_d) \in \{0,1\}^{d \log \lambda}$ be the decomposition of x into substrings of length $\log \lambda$.
2. Parse $k_{\sigma} = s_{\sigma}^{(0)} || \mathsf{CW}^{(1)} || \ldots || \mathsf{CW}^{(d+1)}$ and let $t^0 = \sigma$.
3. For $i = 1$ to d do
   (a) Parse $\mathsf{CW}^{(i)} = s_{\mathsf{CW}}^1 || t_{\mathsf{CW}}^1 || \ldots || s_{\mathsf{CW}}^{\lambda} || t_{\mathsf{CW}}^{\lambda}$.
   (b) $\tau^{(i)} \leftarrow G(s^{(i-1)}) \oplus (t^{(i-1)} \cdot \mathsf{CW}^{(i)})$.
   (c) Parse $\tau^{(i)} \leftarrow s^1 || t^1 || \ldots || s^{\lambda} || t^{\lambda}$.
   (d) Set $s^{(i)} \leftarrow s^{x_i}$ and $t^{(i)} \leftarrow t^{x_i}$.
4. Return $\mathsf{Convert}(s^{(d)}) + t^{(d)} \cdot \mathsf{CW}^{(d+1)} \in \mathbb{F}_p$.

4.2 From N-party DPF to N-party PCG

Starting from the building block of an N-party DPF, it is immediately observed that applying the approach of [BCG+19] results in an N-party PCG for arbitrary constant degree correlations. Based on this PCG, we will derive a single-circuit N-party HSS scheme for constant degree polynomials.

Building on the work of [CM21], our HSS scheme requires a PCG for the *tensor powers* correlation:

Definition 9 (tensor power correlation). *The tensor power correlation is parametrized with a string length n and a (constant) tensor power* tpp. *The correlation generates additive shares of*

$$(1||\mathbf{r})^{\otimes \mathsf{tpp}}, \text{ where } \mathbf{r} \in \mathbb{F}_p^n \text{ is pseudorandom.}$$

It is clear that such a tensor power correlation is a reverse-sampleable correlation as after fixing a set of shares $\{\mathsf{share}_{\sigma}\}_{\sigma \in T}$ for any set $T \subsetneq [N]$ (w.l.o.g, suppose $N \notin T$), the rest of shares can be reverse-sampled by sampling $\mathbf{r}$, randomly sampling $N - |T| - 1$ shares as $\{\mathsf{share}_{\tilde{\sigma}}\}_{\tilde{\sigma} \in [N-1] \setminus T}$, and computing $\mathsf{share}_N \leftarrow (\mathbf{r}^{\otimes j})_{j=1}^{\mathsf{tpp}} - \sum_{\sigma \in [N-1]} \mathsf{share}_{\sigma}$.

To construct a PCG compressing such a correlation, following [BCG+19], the idea is to first distribute constant degree correlation over a sparse vector based on multi-point DPF and then take advantage of the linearity of the dual-LPN assumption to convert this into a constant degree correlation over pseudorandom vectors. Moreover, to satisfy the correctness, we use pairwise PRG seeds to randomize their DPF shares. The security of this construction relies on the dual-LPN assumption, the security of the underlying DPF, along with the security of PRG. We describe the PCG for tensor power correlation in Protocol 2 and state its functionality and property in Theorem 4 below.

Protocol 2: **N-party PCG for the Tensor Power Correlation from N-party DPF**

Parameters: $1^{\lambda}, n, n', t, p, \mathsf{tpp} \in \mathbb{N}$, where $n' > n$. Let $\mathbf{C}$ be a code generation algorithm such that $H_{n',n} \leftarrow_{\$} \mathbf{C}(n', n, \mathbb{F}_p)$ is a random parity-check matrix. Let $\mathsf{PRG} : \{0,1\}^{\lambda} \rightarrow \mathbb{F}_p^{(1+n)^{\mathsf{tpp}}}$ be a PRG.
$\mathsf{PCG.Gen}(1^{\lambda})$:

1. Pick a random t-sparse vector $\mathbf{e} \leftarrow\$ \mathsf{HW}_t(\mathbb{F}_p^{n'})$. Let $f : [(1+n')^{\mathsf{tpp}}] \to \mathbb{F}_p$ be the multi-point function with $(1+t)^{\mathsf{tpp}}$ points, where $f(i)$ returns the i-th coordinate of $(1||\mathbf{e})^{\otimes \mathsf{tpp}}$.
2. Compute $(K_1^{\mathsf{fss}}, K_2^{\mathsf{fss}}, \ldots, K_N^{\mathsf{fss}}) \leftarrow\$ \mathsf{MPFSS}_{\mathsf{tpp}}.\mathsf{Gen}(1^\lambda, f)$.
3. For every $i, j \in [N]$ with $i < j$, sample PRG seeds $s^{ij} \leftarrow\$ \{0,1\}^\lambda$.
4. For all $\sigma \in [N]$, let $k_\sigma \leftarrow (n, K_\sigma^{\mathsf{fss}}, \{s^{j\sigma}\}_{1 \le j < \sigma}, \{s^{\sigma j}\}_{\sigma < j \le N})$.
5. Output $(k_\sigma)_{\sigma \in [N]}$.

$\mathsf{PCG.Expand}(\sigma, k_\sigma)$:

1. On input (σ, k_σ), parse k_σ as $(n, K_\sigma^{\mathsf{fss}}, \{s^{j\sigma}\}_{1 \le j < \sigma}, \{s^{\sigma j}\}_{\sigma < j \le N})$.
2. For every $j \neq \sigma$, compute $t_{\sigma j} = \mathsf{PRG}(s_{\sigma j})$ if $\sigma < j$ and $t_{j\sigma} = \mathsf{PRG}(s_{j\sigma})$ otherwise.
3. Compute $\mathbf{v}_\sigma \leftarrow \mathsf{MPFSS}_{\mathsf{tpp}}.\mathsf{FullEval}(\sigma, K_\sigma^{\mathsf{fss}})$. `\\ Note that` $\mathbf{v}_\sigma \in \mathbb{F}_p^{(1+n')^{\mathsf{tpp}}}$.
4. Output $\mathbf{z}_\sigma \leftarrow (\mathsf{Diag}(1, H_{n',n}))^{\otimes \mathsf{tpp}} \cdot \mathbf{v}_\sigma + \sum_{1 \le j < \sigma} t_{j\sigma} - \sum_{\sigma < j \le M} t_{\sigma j}$. `\\ Note that` $\mathbf{z}_\sigma \in \mathbb{F}_p^{(1+n)^{\mathsf{tpp}}}$ and $(1||\mathbf{r})^{\otimes \mathsf{tpp}} = \sum_{\sigma \in [N]} \mathbf{z}_\sigma$.

Theorem 4. *Let* $\mathsf{DPF}_{\mathsf{tpp}}$ *be an* N*-party distributed point function for the family of point functions* $\{f_{\alpha,\beta} : [(1+n')^{\mathsf{tpp}}] \to \mathbb{F}_p \mid \alpha \in [(1+n')^{\mathsf{tpp}}], \beta \in \mathbb{F}_p\}$ *with key size* ℓ *and* $\mathsf{MPFSS}_{\mathsf{tpp}}$ *be an* N*-party multi-point FSS scheme instantiated with* $(1+t)^{\mathsf{tpp}}$ *invocations of* $\mathsf{DPF}_{\mathsf{tpp}}$ *for some sparseness parameter* t*. Let* $\mathsf{PRG} : \{0,1\}^\lambda \to \mathbb{F}_p^{(1+n)^{\mathsf{tpp}}}$ *be a secure PRG. Suppose the assumption* $\mathbb{F}_p$-$\mathsf{LPN}(n'-n, n', t)$ *holds.*

Then $\mathsf{PCG} = (\mathsf{PCG.Gen}, \mathsf{PCG.Expand})$ *in Protocol 2 is an* N*-party PCG for degree-*tpp *tensor power correlation,* $(1||\mathbf{r})^{\otimes \mathsf{tpp}}$ *where* $\mathbf{r} \in \mathbb{F}_p^n$*, with the following properties.*

- $\mathsf{PCG.Gen}$ *generates* N *seeds, each of size* $(1+t)^{\mathsf{tpp}} \cdot \ell + (N-1) \cdot \lambda$.
- $\mathsf{PCG.Expand}$ *involves* $(1+t)^{\mathsf{tpp}}$ *invocations of* $\mathsf{DPF}_{\mathsf{tpp}}.\mathsf{FullEval}$*,* $N-1$ *invocations of* PRG*, along with* $O((n \cdot n')^{\mathsf{tpp}})$ *arithmetic operations over* $\mathbb{F}_p$*. If* $\mathsf{DPF}_{\mathsf{tpp}}.\mathsf{Eval}$ *can be computed by a polynomial of degree* d*, then* $\mathsf{PCG.Expand}$ *can also be computed by polynomials of degree* d*.*

Proof (Proof of Theorem 4).

Correctness. Let $\mathcal{C}$ denote the degree-tpp tensor power distribution. By correctness of $\mathsf{MPFSS}_{\mathsf{tpp}}$ and multi-linearity of the tensor product, we have that

$$
\begin{aligned}
\sum_{\sigma \in [N]} \mathbf{z}_\sigma &= \sum_{\sigma \in [N]} ((\mathsf{Diag}(1, H_{n',n})^{\otimes \mathsf{tpp}}) \cdot \mathbf{v}_\sigma + \sum_{1 \le j < \sigma} t_{j\sigma} - \sum_{\sigma < j \le M} t_{\sigma j}) \\
&= (\mathsf{Diag}(1, H_{n',n})^{\otimes \mathsf{tpp}}) \cdot \sum_{\sigma \in [N]} \mathbf{v}_\sigma + \sum_{i<j} (t_{ij} - t_{ji}) \\
&= (\mathsf{Diag}(1, H_{n',n})^{\otimes \mathsf{tpp}}) \cdot (1||\mathbf{e})^{\otimes \mathsf{tpp}} = (\mathsf{Diag}(1, H_{n',n}) \cdot (1||\mathbf{e}))^{\otimes \mathsf{tpp}} = (1||\mathbf{r})^{\otimes \mathsf{tpp}}.
\end{aligned}
$$

Then we have

$$
\begin{aligned}
&\left\{ (\mathbf{z}_\sigma)_{\sigma \in [N]} \middle| \begin{array}{l} (k_\sigma)_{\sigma \in [N]} \leftarrow \mathsf{PCG.Gen}(1^\lambda), \\ \mathbf{z}_\sigma \leftarrow \mathsf{PCG.Expand}(\sigma, k_\sigma) \text{ for every } \sigma \in [N] \end{array} \right\} \\
\approx_c &\left\{ (\mathbf{z}_\sigma)_{\sigma \in [N]} \middle| \begin{array}{l} \mathbf{e} \leftarrow\$ \mathsf{HW}_t(\mathbb{F}_p^{n'}), \mathbf{r} \leftarrow H_{n',n} \cdot \mathbf{e}, \\ \text{Sample } (\mathbf{z}_\sigma)_{\sigma \in [N]} \leftarrow\$ (\mathbb{F}_p^{(1+n)^{\mathsf{tpp}}})^N \\ \text{such that } \sum_{\sigma \in [N]} \mathbf{z}_\sigma = (1||\mathbf{r})^{\otimes \mathsf{tpp}} \end{array} \right\} \\
\approx_c &\left\{ (\mathbf{z}_\sigma)_{\sigma \in [N]} \middle| \begin{array}{l} \mathbf{r} \leftarrow\$ \mathbb{F}_p^n, \\ \text{Sample } (\mathbf{z}_\sigma)_{\sigma \in [N]} \leftarrow\$ (\mathbb{F}_p^{(1+n)^{\mathsf{tpp}}})^N \\ \text{such that } \sum_{\sigma \in [N]} \mathbf{z}_\sigma = (1||\mathbf{r})^{\otimes \mathsf{tpp}} \end{array} \right\} = \mathcal{C}(1^\lambda),
\end{aligned}
$$

where the first indistinguishability comes from the security of PRG and the second indistinguishability holds due to the hardness of $\mathbb{F}_p$-$\mathsf{LPN}(n'-n, n', t)$, which completes the PCG correctness of PCG.

Security. Let $T \subsetneq [N]$ denote the set of corrupted parties. By definition of PCG security, we aim to prove given corrupted parties' seeds $(k_\sigma)_{\sigma\in T}$, the distribution of expanded honest parties' shares, $(\mathbf{z}_\sigma)_{\sigma\in T}$, using honest parties' seeds is indistinguishable from that generated by using reverse sampling algorithm, conditioned on expanded corrupted parties' shares, $(\mathbf{z}_\sigma)_{\sigma\in T}$. Formally, the goal is to prove the following two distributions,

$$\mathcal{D}^{\mathrm{real}} \triangleq \left\{ (k_\sigma)_{\sigma\in T}, (\mathbf{z}_{\tilde\sigma})_{\tilde\sigma\in[N]\setminus T} \middle| \begin{array}{l} (k_1,k_2,\ldots,k_N) \leftarrow \mathsf{PCG.Gen}(1^\lambda) \\ (\mathbf{z}_{\tilde\sigma}) \leftarrow \mathsf{PCG.Expand}(\tilde\sigma, k_{\tilde\sigma}) \text{ for every } \tilde\sigma \in [N]\setminus T \end{array} \right\}$$

and

$$\mathcal{D}^{\mathrm{sim}} \triangleq \left\{ (k_\sigma)_{\sigma\in T}, (\mathbf{z}_{\tilde\sigma})_{\tilde\sigma\in[N]\setminus T} \middle| \begin{array}{l} (k_1,k_2,\ldots,k_N) \leftarrow \mathsf{PCG.Gen}(1^\lambda) \\ (\mathbf{z}_\sigma) \leftarrow \mathsf{PCG.Expand}(\sigma, k_\sigma) \text{ for every } \sigma \in T \\ (\mathbf{z}_{\tilde\sigma})_{\tilde\sigma\in[N]\setminus T} \leftarrow \mathsf{RSample}(T, (\mathbf{z}_\sigma)_{\sigma\in T}) \end{array} \right\},$$

are computationally indistinguishable, which will be proved using hybrid arguments.

$\mathbf{Hyb}_0$: In this hybrid, we consider the real world distribution $\mathcal{D}^{\mathrm{real}}$ defined above.

$\mathbf{Hyb}_1$: In this hybrid, instead of computing shares $(\mathbf{z}_{\tilde\sigma})_{\tilde\sigma\in[N]\setminus T}$ of honest parties by expanding their PCG seeds, we compute them by random sampling up to preserving $\sum_{\tilde\sigma\in[N]\setminus T} \mathbf{z}_{\tilde\sigma} = (1||\mathbf{r})^{\otimes \mathsf{tpp}} - \sum_{\sigma\in T} \mathbf{z}_\sigma$. Concretely, we consider the following distribution

$$\mathcal{D}^1 \triangleq \left\{ (k_\sigma)_{\sigma\in T}, (\mathbf{z}_{\tilde\sigma})_{\tilde\sigma\in[N]\setminus T} \middle| \begin{array}{l} (k_1,k_2,\ldots,k_N) \leftarrow \mathsf{PCG.Gen}(1^\lambda), \\ \mathbf{z}_\sigma \leftarrow \mathsf{PCG.Expand}(\sigma, k_\sigma) \text{ for every } \sigma \in T, \\ \text{Sample } (\mathbf{z}_{\tilde\sigma})_{\tilde\sigma\in[N]\setminus T} \leftarrow\$ (\mathbb{F}_p^{(1+n)^{\mathsf{tpp}}})^{N-|T|} \\ \text{such that } \sum_{\tilde\sigma\in[N]\setminus T} \mathbf{z}_{\tilde\sigma} = (1||\mathbf{r})^{\otimes\mathsf{tpp}} - \sum_{\sigma\in T} \mathbf{z}_\sigma \end{array} \right\},$$

where $\mathbf{r}$ is computed inside $\mathsf{PCG.Gen}(1^\lambda)$. The distributions of $\mathcal{D}^1$ and $\mathcal{D}^{\mathrm{real}}$ are computationally indistinguishable because of the security of PRG.

$\mathbf{Hyb}_2$: In this hybrid, instead of invoking $\mathsf{PCG.Gen}(1^\lambda)$, we compute $(K_\sigma^{\mathsf{fss}})_{\sigma\in T}$ using the MPFSS simulator $\mathsf{Sim}(1^\lambda, T, \cdots)$. Concretely, we consider the following distribution

$$\mathcal{D}^2 \triangleq \left\{ \begin{array}{l} (k_\sigma)_{\sigma\in T}, \\ (\mathbf{z}_{\tilde\sigma})_{\tilde\sigma\in[N]\setminus T} \end{array} \middle| \begin{array}{l} (K_\sigma^{\mathsf{fss}})_{\sigma\in T} \leftarrow \mathsf{Sim}(1^\lambda, T, \cdots), \\ s^{ij} \leftarrow\$ \{0,1\}^\lambda \text{ for every } i,j\in[N] \text{ with } i<j, (i\in T)\vee(j\in T), \\ k_\sigma \leftarrow (n, K_\sigma^{\mathsf{fss}}, \{s^{j\sigma}\}_{1\le j<\sigma}, \{s^{\sigma j}\}_{\sigma<j\le N}) \text{ for every } \sigma\in T, \\ \mathbf{z}_\sigma \leftarrow \mathsf{PCG.Expand}(\sigma, k_\sigma) \text{ for every } \sigma \in T, \\ \mathbf{e} \leftarrow\$ \mathsf{HW}_t(\mathbb{F}_p^{n'}), \mathbf{r} \leftarrow H_{n',n}\cdot\mathbf{e}, \\ \text{Sample } (\mathbf{z}_{\tilde\sigma})_{\tilde\sigma\in[N]\setminus T} \leftarrow\$ (\mathbb{F}_p^{(1+n)^{\mathsf{tpp}}})^{N-|T|} \\ \text{such that } \sum_{\tilde\sigma\in[N]\setminus T} \mathbf{z}_{\tilde\sigma} = (1||\mathbf{r})^{\otimes\mathsf{tpp}} - \sum_{\sigma\in T} \mathbf{z}_\sigma \end{array} \right\}.$$

The distributions of $\mathcal{D}^2$ and $\mathcal{D}^1$ are computationally indistinguishable due to the security of MPFSS.

$\mathbf{Hyb}_3$: In this hybrid, instead of computing $\mathbf{r}$ by compressing a sparse vector, we randomly sample $\mathbf{r}' \leftarrow\$ \mathbb{F}_p^n$. Concretely, we consider the following distribution

$$\mathcal{D}^3 \triangleq \left\{ \begin{array}{l} (k_\sigma)_{\sigma\in T}, \\ (\mathbf{z}_{\tilde\sigma})_{\tilde\sigma\in[N]\setminus T} \end{array} \middle| \begin{array}{l} (K_\sigma^{\mathsf{fss}})_{\sigma\in T} \leftarrow \mathsf{Sim}(1^\lambda, T, \cdots), \\ s^{ij} \leftarrow\$ \{0,1\}^\lambda \text{ for every } i,j\in[N] \text{ with } i<j, (i\in T)\vee(j\in T), \\ k_\sigma \leftarrow (n, K_\sigma^{\mathsf{fss}}, \{s^{j\sigma}\}_{1\le j<\sigma}, \{s^{\sigma j}\}_{\sigma<j\le N}) \text{ for every } \sigma\in T, \\ \mathbf{z}_\sigma \leftarrow \mathsf{PCG.Expand}(\sigma, k_\sigma) \text{ for every } \sigma \in T, \\ \mathbf{r} \leftarrow\$ \mathbb{F}_p^n, \\ \text{Sample } (\mathbf{z}_{\tilde\sigma})_{\tilde\sigma\in[N]\setminus T} \leftarrow\$ (\mathbb{F}_p^{(1+n)^{\mathsf{tpp}}})^{N-|T|} \\ \text{such that } \sum_{\tilde\sigma\in[N]\setminus T} \mathbf{z}_{\tilde\sigma} = (1||\mathbf{r})^{\otimes\mathsf{tpp}} - \sum_{\sigma\in T} \mathbf{z}_\sigma \end{array} \right\}.$$

The distributions of $\mathcal{D}^3$ and $\mathcal{D}^2$ are computationally indistinguishable due to the hardness of $\mathbb{F}_p$-$\mathsf{LPN}(n'-n, n', t)$.

Hyb$_4$: In this hybrid, instead of computing $(k_\sigma)_{\sigma\in T}$ by using $\mathsf{Sim}(1^\lambda, T, \cdots)$, we compute them by invoking an independent instance of $\mathsf{PCG.Gen}$. Concretely, we consider the following distribution

$$\mathcal{D}^4 \triangleq \left\{ (k_\sigma)_{\sigma\in T}, (\mathbf{z}_{\tilde\sigma})_{\tilde\sigma\in[N]\setminus T} \middle| \begin{array}{l} (k_1, k_2, \ldots, k_N)_{\sigma\in T} \leftarrow \mathsf{PCG.Gen}(1^\lambda), \\ \mathbf{z}_\sigma \leftarrow \mathsf{PCG.Expand}(\sigma, k_\sigma) \text{ for every } \sigma\in T, \\ \mathbf{r} \leftarrow\$\ \mathbb{F}_p^n, \\ \text{Sample } (\mathbf{z}_{\tilde\sigma})_{\tilde\sigma\in[N]\setminus T} \leftarrow\$\ (\mathbb{F}_p^{(1+n)^{\mathsf{tpp}}})^{N-|T|} \\ \text{such that } \sum_{\tilde\sigma\in[N]\setminus T} \mathbf{z}_{\tilde\sigma} = (1||\mathbf{r})^{\otimes \mathsf{tpp}} - \sum_{\sigma\in T} \mathbf{z}_\sigma \end{array} \right\}.$$

The distributions of $\mathcal{D}^4$ and $\mathcal{D}^3$ are computationally indistinguishable due to the security of MPFSS.

Since the distributions of $\mathcal{D}^4$ and $\mathcal{D}^{\text{sim}}$ are identical, we prove the PCG security of PCG. The properties of PCG follow immediately from the construction. □

4.3 From N-party PCG to N-party HSS

Also observed in [BCG+19], starting from an N-party PCG for constant degree polynomial correlations, one can construct an N-party HSS which allows to distributively evaluate constant degree polynomials. In particular, given a public vector $\mathbf{P} : \mathbb{F}_p^n \to \mathbb{F}_p^m$ of m degree-d polynomials in n variables and shares of a secret input $\mathbf{x}\in\mathbb{F}_p^n$, all parties aim to compute their shares of $\mathbf{P}(\mathbf{x})$ by locally evaluating $\mathbf{P}$ on their HSS shares.

The idea is that all parties first invoke PCG for degree-d tensor power correlation to obtain their shares of $(1||\mathbf{r})^{\otimes d}$ with a pseudorandom $\mathbf{r}$. For each party, include $\mathbf{x}' \leftarrow \mathbf{x}+\mathbf{r}$ together with its PCG seed as its HSS share. Define $\hat{\mathbf{P}}(\cdot)$ such that $\hat{\mathbf{P}}(\mathbf{X}) = \mathbf{P}(\mathbf{X}-\mathbf{r})$. As a result, all parties additively share the coefficients of $\hat{\mathbf{P}}(\cdot)$, which are public linear combinations of elements in $(1||\mathbf{r})^{\otimes d}$. Then all parties can locally compute their additive shares of $\mathbf{P}(\mathbf{x})$ based on their HSS shares by observing that $\mathbf{P}(\mathbf{x}) = \mathbf{P}(\mathbf{x}'-\mathbf{r}) = \hat{\mathbf{P}}(\mathbf{x}')$ with $\mathbf{x}'$ known to all parties.

The security of such an HSS scheme stems out from the security of the underlying PCG for tensor power correlations. In more details, given corrupted parties' PCG seeds, $\mathbf{r}$ is still pseudorandom so that the secrecy of the input $\mathbf{x}$ can be protected. We present the construction in Protocol 3 and conclude its functionality and properties in Theorem 5.

Protocol 3: **N-party HSS from N-party PCG**

Parameters: $1^\lambda, n, p \in \mathbb{N}$. Let $\mathbf{P} : \mathbb{F}_p^n \to \mathbb{F}_p^m$ be a public vector of m degree-d polynomials in n variables.

$\mathsf{HSS.Share}(1^\lambda, \mathbf{x})$:

1. Compute $(k_\sigma)_{\sigma\in[N]} \leftarrow \mathsf{PCG.Gen}(1^\lambda)$. `\\ Note that` k_σ `expands to additive shares of` $(1||\mathbf{r})^{\otimes d}$, `where` $\mathbf{r}\in\mathbb{F}_p^n$ `is pseudorandom.`
2. Set $\mathbf{x}' \leftarrow \mathbf{x}+\mathbf{r}$ and $s_\sigma \leftarrow (k_\sigma, \mathbf{x}')$ for $\sigma\in[N]$.
3. Output $\{s_\sigma\}_{\sigma\in[N]}$.

$\mathsf{HSS.Eval}(\sigma, s_\sigma, \mathbf{P})$: on input party index $\sigma\in[N]$, share s_σ, and a vector $\mathbf{P}$ of m degree-d polynomials in n variables, compute an additive share sh_σ of $P(\mathbf{x})$.

1. Parse s_σ as $(k_\sigma, \mathbf{x}')$.
2. Compute σ-th additive share sh_σ of $\mathbf{P}(\mathbf{x}'-\mathbf{r})$ via $\mathsf{PCG.Expand}(\mathsf{œ}, \mathsf{k_œ})$ and $\mathbf{x}'$. `\\ Recall` $\mathbf{P}(\mathbf{x}'-\mathbf{r}) = \mathbf{P}(\mathbf{x})$ `can be viewed as a degree-`d `multivariate polynomial in` $\mathbf{x}'$ `with coefficients being linear combinations of elements in` $(1||\mathbf{r})^{\otimes d}$ `which are additively shared by all parties.`

Theorem 5. *Let* $\mathsf{PCG} = (\mathsf{PCG.Gen}, \mathsf{PCG.Expand})$ *be an N-party PCG for degree-d tensor powers correlation* $(1||\mathbf{r})^{\otimes d}$ *where* $\mathbf{r}$ *is pseudorandom and output seeds with each of size* ℓ.

Then $\mathsf{HSS} = (\mathsf{HSS.Share}, \mathsf{HSS.Eval})$ *in Protocol 3 is a secure N-party, single-circuit HSS scheme for general degree-d multivariate polynomials over $\mathbb{F}_p$ with the following properties.*

– $\mathsf{HSS.Share}$ *outputs N shares, with each of size $n + \ell$.*
– $\mathsf{HSS.Eval}$ *can be computed by a polynomial of degree D if* $\mathsf{PCG.Expand}$ *can be computed by a polynomial of degree D.*

Proof (Proof of Theorem 5). As for the correctness, by correctness of PCG, we have that $\sum_{\sigma\in[N]} \mathsf{sh}_\sigma = \mathbf{P}(\mathbf{x}' - \mathbf{r}) = \mathbf{P}(\mathbf{x})$. We then consider the security. For any set $T \subsetneq [N]$ of corrupted parties, for any secret input $\mathbf{x} \in \mathbb{F}_p^n$, the real world distribution is as below

$$\mathcal{D}^{\mathrm{real}} \triangleq \left\{ (s_\sigma)_{\sigma\in T} \,\middle|\, (s_1, s_2, \ldots, s_N) \leftarrow \mathsf{HSS.Share}(1^\lambda, \mathbf{x}) \right\}.$$

We construct a *PPT* simulator $\mathsf{Sim}(1^\lambda, 1^N, 1^n)$ which works as follows.

– Sim invokes PCG to generate the seeds: $(k_1, k_2, \ldots, k_N) \leftarrow \mathsf{PCG.Gen}(1^\lambda)$.
– Randomly sample $\mathbf{x}' \leftarrow\$ \mathbb{F}_p^n$.
– For every $\sigma \in T$, set $s_\sigma \leftarrow (k_\sigma, \mathbf{x}')$. Output $(s_\sigma)_{\sigma\in T}$.

We then prove the effectiveness of the simulator using the following hybrid arguments.

$\mathbf{Hyb}_0$: In this hybrid, we consider the real world distribution $\mathcal{D}^{\mathrm{real}}$ defined above and notice that

$$\mathcal{D}^{\mathrm{real}} \equiv \left\{ (s_\sigma)_{\sigma\in T} \,\middle|\, \begin{array}{l} (k_1, k_2, \ldots, k_N) \leftarrow \mathsf{PCG.Gen}(1^\lambda), \\ \mathbf{z}_\ell \leftarrow \mathsf{PCG.Expand}(\ell, k_\ell) \text{ for every } \ell \in [N], \\ (1||\mathbf{r})^{\otimes d} \leftarrow \sum_{\ell\in[N]} \mathbf{z}_\ell, \\ \mathbf{x}' \leftarrow \mathbf{x} + \mathbf{r}, \\ s_\sigma \leftarrow (k_\sigma, \mathbf{x}') \text{ for every } \sigma \in T \end{array} \right\}.$$

$\mathbf{Hyb}_1$: In this hybrid, instead of computing $\mathbf{r}$ from expanded tensor power correlation using PCG seeds, we randomly sample $\mathbf{r} \leftarrow\$ \mathbb{F}_p^n$. Concretely, we consider the following distribution

$$\mathcal{D}^1 \triangleq \left\{ (s_\sigma)_{\sigma\in T} \,\middle|\, \begin{array}{l} (k_1, k_2, \ldots, k_N) \leftarrow \mathsf{PCG.Gen}(1^\lambda), \\ \mathbf{z}_\sigma \leftarrow \mathsf{PCG.Expand}(\sigma, k_\sigma) \text{ for every } \sigma \in T, \\ (\mathbf{z}_{\tilde\sigma})_{\tilde\sigma\in[N]\setminus T} \leftarrow \mathsf{RSample}(T, (\mathbf{z}_\sigma)_{\sigma\in T}), \\ (1||\mathbf{r})^{\otimes d} \leftarrow \sum_{\ell\in[N]} \mathbf{z}_\ell, \\ \mathbf{x}' \leftarrow \mathbf{x} + \mathbf{r}, \\ s_\sigma \leftarrow (k_\sigma, \mathbf{x}') \text{ for every } \sigma \in T \end{array} \right\}$$

$$\equiv \left\{ (s_\sigma)_{\sigma\in T} \,\middle|\, \begin{array}{l} (k_1, k_2, \ldots, k_N) \leftarrow \mathsf{PCG.Gen}(1^\lambda), \\ \mathbf{r} \leftarrow\$ \mathbb{F}_p^n, \mathbf{x}' \leftarrow \mathbf{x} + \mathbf{r}, \\ s_\sigma \leftarrow (k_\sigma, \mathbf{x}') \text{ for every } \sigma \in T \end{array} \right\}.$$

The distributions of $\mathcal{D}^1$ and $\mathcal{D}^{\mathrm{real}}$ are computationally indistinguishable due to the PCG security of PCG.

$\mathbf{Hyb}_2$: In this hybrid, instead of computing $\mathbf{x}'$ as $\mathbf{x} + \mathbf{r}$ with $\mathbf{r} \in \mathbb{F}_p^n$ sampled uniformly at random, we directly sample $\mathbf{x}'$ uniformly at random. Concretely, we consider the following distribution

$$\mathcal{D}^2 \triangleq \left\{ (s_\sigma)_{\sigma\in T} \,\middle|\, \begin{array}{l} (k_1, k_2, \ldots, k_N) \leftarrow \mathsf{PCG.Gen}(1^\lambda), \\ \mathbf{x}' \leftarrow\$ \mathbb{F}_p^n, \\ s_\sigma \leftarrow (k_\sigma, \mathbf{x}') \text{ for every } \sigma \in T \end{array} \right\}.$$

The distributions of $\mathcal{D}^2$ and $\mathcal{D}^1$ are identical because $\mathbf{r}$ is sampled uniformly at random and unkown to the adversary.

Since $\mathbf{Hyb}_2$ already generates the distribution the same way as Sim does, we prove the HSS security of HSS. Two properties of HSS immediately follow from our construction. ◻

4.4 From N-party DPF and M-party HSS to $N \cdot M$-party DPF

Following [CK24], we apply the nesting approach to construct an $N \cdot M$-party DPF from an N-party DPF and an M-party HSS allowing to homomorphically evaluate a class of functions that includes the evaluation algorithm of the N-party DPF. In particular, we first apply the N-party DPF to generate N DPF keys that allow to locally evaluate the secret point function and then make use of the M-party HSS to secret share each DPF key into M shares. The security of the resulting $N \cdot M$-party DPF comes from the security of N-party DPF as well as the security of M-party HSS.

We present the construction in Protocol 4 and conclude its functionality and properties in Theorem 6 (Theorem 12 in [CK24]).

Protocol 4: **NM-party DPF from N-party DPF and M-party HSS**

Parameters: Let $\mathsf{DPF} = (\mathsf{DPF.Gen}, \mathsf{DPF.Eval})$ be an N-party DPF with domain $[D]$, output group $\mathbb{G}$, and share size ℓ. Let $\mathsf{HSS} = (\mathsf{HSS.Share}, \mathsf{HSS.Eval})$ be an M-party HSS for a class of functions $\mathcal{F}$ such that for every $x \in [D]$, $i \in [N]$, the function $f_{i,x} : s \mapsto \mathsf{DPF.Eval}(i, s, x)$ belongs to $\mathcal{F}$.
$\mathsf{DPF}^*.\mathsf{Gen}(1^\lambda, \alpha, \beta, \mathbb{G})$: Let $(s_1, \ldots, s_N) \leftarrow \mathsf{DPF.Gen}(1^\lambda, \alpha, \beta, \mathbb{G})$. Then set $(k_{i,1}, \ldots, k_{i,M}) \leftarrow \mathsf{HSS.Share}(1^\lambda, s_i)$ for $i \in [N]$ and output $(k_{i,j})_{i\in[N],j\in[M]}$.
$\mathsf{DPF}^*.\mathsf{Eval}((i,j), k_{i,j}, x)$: Define $f_{i,x} : s \mapsto \mathsf{DPF.Eval}(i, s, x)$. Output $\mathsf{HSS.Eval}(j, f_{i,x}, k_{i,j})$.

Theorem 6. *Let* $\mathsf{DPF} = (\mathsf{DPF.Gen}, \mathsf{DPF.Eval})$ *be an N-party DPF for the family of point functions* $\{f_{\alpha,\beta} : [D] \to \mathbb{F}_p \mid \alpha \in [D], \beta \in \mathbb{F}_p\}$ *such that* $\mathsf{DPF.Eval}$ *can be computed by a constant-degree polynomial. Let* $\mathsf{HSS} = (\mathsf{HSS.Share}, \mathsf{HSS.Eval})$ *be an M-party HSS scheme with share size ℓ for a class of functions $\mathcal{F}$ such that for every $x \in [D], i \in [N]$, the function* $f_{i,x} : s \mapsto \mathsf{DPF.Eval}(i, s, x)$ *belongs to $\mathcal{F}$.*

Then the scheme $\mathsf{DPF}^* = (\mathsf{DPF}^*.\mathsf{Gen}, \mathsf{DPF}^*.\mathsf{Eval})$ *from Protocol 4 is an $N \cdot M$-party DPF for the family of point functions* $\{f_{\alpha,\beta} : [D] \to \mathbb{F}_p \mid \alpha \in [D], \beta \in \mathbb{F}_p\}$ *with the following properties.*

- $\mathsf{DPF}^*.\mathsf{Gen}$ *outputs $N \cdot M$ keys, each of size ℓ.*
- $\mathsf{DPF}^*.\mathsf{Eval}$ *viewed as a function with the DPF key as its input can be evaluated by the class $\mathcal{F}^*$ of functions that includes* $\{s \mapsto \mathsf{HSS.Eval}(j, g, s) \mid j \in [M], g \in \mathcal{F}\}$.

5 Multiparty DPF and HSS via Iterative Nesting

Using the building blocks overviewed in section 4, we present our main results: an iterative nesting approach to construct 2^m-party DPF, HSS, and PCG for any constant m, and a boosting approach to boost the number of parties to an arbitrary polynomial N.

5.1 Towards 2^m-party DPF via iterative nesting

In this section we outline how the previous results can be combined to yield a 2^m-party DPF, starting from the building block of a 2-party DPF, instantiating the nesting technique of [CK24] as outlined in section 4.4. While [CK24] instantiated this template using HSS for log-depth circuits from a variety of assumptions (including DDH, DCR and LWE), our following scheme utilizes HSS for *constant*-depth circuits constructed via a 2^{m-1}-party DPF and the LPN assumption with constant number of samples. For starters, we see how the technique so described can be used to construct a 4-party DPF.

- From Theorem 3, we get a 2-party distributed point function DPF for functions with domain size $[\lambda^d]$ whose key is of $O(d \cdot \lambda^2)$ bits and and whose evaluation function can be computed by degree-c^d polynomials. As per our nesting strategy, the first step is to use a 2-party HSS scheme to create *HSS shares* of the two *DPF shares* (k_0, k_1) output by $\mathsf{DPF.Eval}$. To do this, we require an HSS scheme that supports the evaluation of circuits that can be computed by degree-c^d polynomials.

– We can iteratively construct such an HSS scheme for degree-c^d polynomials using our template. To do this, we start by constructing a 2-party PCG for the tensor powers correlation with $\mathsf{tpp} = c^d$ as per Theorem 4. Our PCG outputs shares of all tensor powers of a length-n pseudorandom vector $\mathbf{r}$. Since we wish to obtain HSS shares of inputs to $\mathsf{DPF.Eval}$, we set $n = O(\lambda^2)$. Such a PCG requires a second DPF (which we term DPF') with domain $[(n')^{\mathsf{tpp}}]$. Setting $n = O(n')$ (which corresponds to LPN with linear number of samples), it follows that the domain of DPF' is $[O(\lambda^2)^{c^d}]$, which is still polynomial in λ. As per Theorem 3, DPF' can be evaluated by a polynomial of degree $\approx c^{2c^d}$. Hence, from Theorem 4, assuming $\mathbb{F}\text{-}\mathsf{LPN}(O(\lambda^2), O(\lambda^2), t)$, we obtain a 2-party PCG for the tensor powers correlation whose key size is $O(t^{2c^d} \cdot \lambda^2)$ and whose expand function involves $O(t^{2c^d} \cdot \lambda^{2c^d})$ PRF calls and $O((n \cdot n')^{c^d})$ arithmetic computations. Furthermore, this PCG can be evaluated by a polynomial of degree (close to) c^{2c^d}.
– Based on Theorem 5, the previous PCG gives us a 2-party HSS for degree-d polynomials with n variables whose share size is $O(t^{2c^d} \cdot \lambda^2)$ and its evaluation function is computable by a degree-c^{2c^d} polynomial.
– Combining the above with Theorem 6, we obtain a 4-party DPF for point functions with domain size $[\lambda^d]$, key size $O(t^{2c^d} \cdot \lambda^2)$ bits and evaluation function computable by degree-c^{2c^d} polynomials.

Note that the above technique can be applied using any M-party HSS scheme to yield a $2M$-party DPF. By Theorem 6, it follows that both the share size and the computational complexity of $\mathsf{DPF.Eval}$ for the resultant DPF is inherited from the M-party HSS scheme.

5.2 Full construction

We now state an analyze the inductive construction sketched in the previous section.

Theorem 7. *Fix integers $m, d > 0$. Assume the existence of a pseudorandom generator $G : \{0,1\}^\lambda \to \{0,1\}^{\lambda\cdot(\lambda+1)}$ that can be computed by a degree-c polynomial and define $d_{2^m,d} = (1/3) \cdot (3, c, d) \uparrow m$. Then assuming in addition the dual $\mathbb{F}\text{-}\mathsf{LPN}(\lambda^3/2+1, \lambda^3, \lambda^{1/(2d_{2^{m-1},d})})$ assumption, there exists a secure, 2^m-party distributed point function DPF that supports the family of point functions $f_{\alpha,\beta} : [\lambda^d] \to \mathbb{F}_p$ with the following characteristics:*

- *The key size is $\ell_{2^m,d} = O(\lambda^3)$.*
- $\mathsf{DPF.Gen}$ *can be computed using $6d_{2^m,d}$ calls to G and $3\lambda^2 \cdot d_{2^m,d} + m \cdot \mathsf{poly}(\lambda)$ additional Boolean operations (where the exponent in poly is independent of c, d, m).*
- *For any input $x \in [\lambda^d]$, $\mathsf{DPF.Eval}(\cdot, x)$ can be computed by a degree-$d_{2^m,d}$ polynomial.*

Proof. Our construction follows the outline provided in the section above. Let $\mathsf{DPF} = (\mathsf{DPF.Gen}, \mathsf{DPF.Eval})$ be the low-complexity distributed point function of Protocol 3 that supports the family of point functions $f_{\alpha,\beta} : [\lambda^d] \to \mathbb{F}_p$ and has key size $\ell_{2,d} = O(d \cdot \lambda^2) \leq \lambda^3/2 - 1$ (where the inequality holds for any constant d, for a large enough λ). The evaluation function of this DPF can be computed by a degree-c^d polynomial. We will prove the above proposition by induction. It is easy to see that DPF satisfies the proposition for $m = 1$.

In order to obtain a 2^{m+1}-party DPF, we begin by constructing a 2^m-party HSS scheme for degree-$c^d = d_{2,d}$ polynomials. Inductively, let $\mathsf{DPF}_{2^m,3c^d}$ be a 2^m-party distributed point function that supports the family of point functions $f_{\alpha,\beta} : [\lambda^{3c^d}] \to \mathbb{F}_p$ which has key size $\ell_{2^m,3c^d}$ and can be evaluated by a degree-$d_{2^m,3c^d}$ polynomial. By Theorem 4, assuming the hardness of the $\mathbb{F}_p\text{-}\mathsf{LPN}(\lambda^3/2+1, \lambda^3, t_{2^m,3c^d} = \lambda^{1/(2^m \cdot d_{2,d})} - 1)$ and using the fact that $2\ell_{2,d} + 1 \leq \lambda^3/2$, $\mathsf{DPF}_{2^m,3c^d}$ can be bootstrapped into a PCG for degree-c^d $\ell_{2,d}$-variate correlations with seeds of size at most $\lambda^{1/2^m} \cdot \ell_{2^m,3c^d}$ (using $\lambda = (1+t)^{c^d}$) and that can be evaluated by a degree-$d_{2^m,3c^d}$ polynomial. Then, Theorem 5 then immediately implies a 2^m-party HSS scheme for $\ell_{2,d}$-variate polynomials of degree-c^d with share size $\ell_{2,d} + \lambda^{1/2^m} \cdot \ell_{2^m,3c^d}$ such that $\mathsf{HSS.Eval}$ can be computed by a degree-$d_{2^m,3c^d}$ polynomial.

Now, using Theorem 6, we obtain a 2^{m+1}-party DPF $\mathsf{DPF}^* = (\mathsf{DPF}^*.\mathsf{Gen}, \mathsf{DPF}^*.\mathsf{Eval})$ over the domain $[\lambda^d]$ with key size $\ell_{2^{m+1},d} = \ell_{2,d} + \lambda^{1/2^m} \cdot \ell_{2^m,3c^d}$ and such that for each $i \in [2^{m+1}]$ and each $x \in [\lambda^d]$ the function $k \mapsto \mathsf{DPF}^*.\mathsf{Eval}(i, k, x)$ belongs to the family of degree-$d_{2^m,3c^d}$ polynomials. We therefore obtain the following inductive relations:

– $\ell_{2^{m+1},d} = \ell_{2,d} + \lambda^{1/2^m} \cdot \ell_{2^m,3c^d}$ with $\ell_{2,d} = d\lambda^2$,
– $d_{2^{m+1},d} = d_{2^m,3c^d}$ with $d_{2,d} = c^d$.

We show that there exists a constant $\gamma(c,d,m)$ such that $\ell_{2^m,d} \le \gamma(c,d,m)\cdot\lambda^{\sum_{i=0}^{m-1} 1/2^i+1} \le \gamma(c,d,m)\cdot\lambda^3$. Indeed, we have $\ell_{2,d} = \gamma(c,d,1)\cdot\lambda^2$ by defining $\gamma(c,d,1) := d$, and by induction, if $\ell_{2^m,d} \le \gamma(c,d,m)\cdot\lambda^{\sum_{i=0}^{m-1} 1/2^i+1}$, then

$$\begin{aligned}\ell_{2^{m+1},d} &= \ell_{2,d} + \lambda\cdot\ell_{2^m,3c^d} \le d\lambda^2 + \lambda^{1/2^m}\cdot\gamma(c,3c^d,m)\cdot\lambda^{\sum_{i=0}^{m-1} 1/2^i+1}\\ &\le \gamma(c,d,m+1)\cdot\lambda^{\sum_{i=0}^{m} 1/2^i+2},\end{aligned}$$

by setting for instance $\gamma(c,d,m+1) := d+\gamma(c,3c^d,m)$. For $d_{2^m,d}$, we obtain a closed form using the iterated exponential notation:

$$d_{2^m,d} = \frac{(3,c,d)\uparrow m}{3}.$$

We now turn to the complexity of $\mathsf{DPF}^*.\mathsf{Gen}$. Let $c_{2^m,d}$ denote the cost of $\mathsf{DPF}_{2^m,d}$ (counted as a number of calls to G and a number of additional Boolean operations). By Theorem 4, we build a 2^m-party PCG using $(t_{2^m,3c^d}+1)^{3c^d} = \lambda^{1/2^m}$ calls to $\mathsf{DPF}_{2^m,d}$. The conversion to a 2^m-party HSS only incurs an additional $\mathsf{poly}(\ell_{2,d}) = \mathsf{poly}(\lambda)$ cost for stretching $\boldsymbol{r} \in \mathbb{F}^{\ell_{2,d}}$ and computing $\boldsymbol{x}+\boldsymbol{r}$, and $\mathsf{DPF}_{2^{m+1},d}$ is obtained by running $\mathsf{HSS.Share}$ on the keys of DPF. Therefore, we have

$$c_{2^{m+1},d} = \lambda^{1/2^m}\cdot c_{2^m,3c^d} + \mathsf{poly}(\lambda).$$

Furthermore, for $m=1$, $c_{2,d}$ is the cost of DPF given by theorem 3: $c_{2,d} = (2d, O(d\lambda^2))$ ($2d$ invocations of G and $O(d\lambda^2)$ additional Boolean operations). Solving the inductive relation yields

$$c_{2^m,d} = (6d_{2^m,d}, \lambda^2\cdot d_{2^m,d} + m\cdot\mathsf{poly}(\lambda)).$$

It remains to pinpoint the flavor of LPN used in the construction. In short, we proved that if (1) there exists a 2^m-party DPF over a domain of size λ^{3c^d} and (2) the $\mathbb{F}\text{-}\mathsf{LPN}(\lambda^3/2+1, \lambda^3, t_{2^m,3c^d} = \lambda^{1/(2^m\cdot d_{2,d})}-1)$ assumption holds, then there exists a 2^{m+1}-party DPF over a domain of size λ^d. By induction, we build a 2^{m+1}-party DPF from the existence of a 2-party DPF over a domain of size $\lambda^{(c,d)\Uparrow m}$ assuming the hardness of $\mathbb{F}\text{-}\mathsf{LPN}(\lambda^3/2+1, \lambda^3, \lambda^{1/(2^{m-j}\cdot d_{2^j,d})}-1)$ for $j=1$ to m. All these assumptions are implied by $\mathbb{F}\text{-}\mathsf{LPN}(\lambda^3/2+1, \lambda^3, \lambda^{1/(2d_{2^m,d})}-1)$ (as $2^{m-(j+1)}d_{2^{j+1},d} \gg 2^{m-j}d_{2^j,d}$), which concludes the proof.

5.3 Boosting the number of parties

Observe that since we are only able to derive a multiparty DPF within a constant number of parties, using the technique described in section 4.2 can merely result in PCG among a constant number of parties. To obtain a PCG for a polynomial number of parties, we apply a general transformation that originally appears in [BCG+19,BCG+20] which is used to build an M-party PCG for bilinear correlations upon a 2-party PCG for bilinear correlations that satisfies an additional *programmability* property. The idea is that to additively share all the "cross terms", we need to execute $M\cdot(M-1)$ pairwise instances of the underlying 2-party PCGs. Importantly, to guarantee the consistency which allows each party to obtain the same share among different 2-party PCG instances, we further require the base 2-party PCG to satisfy the so-called programmability property, making it possible to "reuse" the inputs without compromising security. Adapting to our application, we formally define the programmable property of multiparty PCGs for constant-degree correlations as follows.

Definition 10 (Programmable PCG). *A tuple of algorithms* $\mathsf{PCG} = (\mathsf{PCG.Gen}, \mathsf{PCG.Expand})$ *following the syntax of a standard PCG, but where* $\mathsf{PCG.Gen}(1^\lambda)$ *takes additional random inputs* $(\rho_1, \rho_2, \ldots, \rho_d)$ *with* $\rho_\sigma \in \{0,1\}^*$ *for each* $\sigma \in [d]$*, is a d-party* programmable *PCG for degree-d correlation denoted by* $\mathcal{C}_d^n$*, where each party* P_σ *takes* $\mathbf{R}_\sigma \in \mathbb{F}_p^n$ *and* $\mathbf{S}_\sigma \in \mathbb{F}_p^{n^d}$*, with* $\mathbf{S}_\sigma$ *denoting the* σ*-th additive share of the tensor product vector* $\bigotimes_{i\in[d]}\mathbf{R}_i \in \mathbb{F}_p^{n^d}$*, if the following holds:*

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

$$\left\{ ((\mathbf{R}_1,\mathbf{S}_1),\ldots,(\mathbf{R}_d,\mathbf{S}_d)) \;\middle|\; \begin{array}{r} \rho_1,\ldots,\rho_d \leftarrow_\$ \{0,1\}^* \\ (k_\sigma)_{\sigma\in[d]} \leftarrow_\$ \mathsf{PCG.Gen}(1^\lambda,\rho_1,\ldots,\rho_d) \\ (\mathbf{R}_\sigma,\mathbf{S}_\sigma) \leftarrow_\$ \mathsf{PCG.Expand}(\sigma,k_\sigma) \textit{ for every } \sigma\in[d] \end{array} \right\}$$

is computationally indistinguishable from $\mathcal{C}_d^n(1^\lambda)$.

– ***Programmability***. *There exist public efficiently computable functions* $\phi_\sigma : \{0,1\}^* \to \mathbb{F}_p^n$ *for each* $\sigma\in[d]$ *such that*

$$\Pr\left[\begin{array}{c} \rho_1,\ldots,\rho_d \leftarrow_\$ \{0,1\}^*, \\ (k_1,\ldots,k_d) \leftarrow_\$ \mathsf{PCG.Gen}(1^\lambda,\rho_1,\ldots,\rho_d), \\ (\mathbf{R}_1,\mathbf{S}_1) \leftarrow \mathsf{PCG.Expand}(1,k_1), \\ \vdots \\ (\mathbf{R}_d,\mathbf{S}_d) \leftarrow \mathsf{PCG.Expand}(d,k_d) \end{array} : \begin{array}{c} \mathbf{R}_1 = \phi_1(\rho_1), \\ \vdots \\ \mathbf{R}_d = \phi_d(\rho_d) \end{array} \right] \geq 1 - \mathsf{negl}(\lambda).$$

– ***Programmable Security***. *For any* $T \subsetneq [d]$, *the distributions*

$$\left\{ (\{k_\sigma\}_{\sigma\in T},(\rho_1,\ldots,\rho_d)) \;\middle|\; \begin{array}{l} (\rho_i)_{i\in[d]} \leftarrow_\$ \{0,1\}^* \\ (k_i)_{i\in[d]} \leftarrow_\$ \mathsf{PCG.Gen}(1^\lambda,\rho_1,\ldots,\rho_d) \end{array} \right\}$$

and

$$\left\{ (\{k_\sigma\}_{\sigma\in T},(\rho_1,\ldots,\rho_d)) \;\middle|\; \begin{array}{l} (\rho_i)_{i\in[d]}, (\tilde\rho_{\tilde\sigma})_{\tilde\sigma\in[d]\setminus T} \leftarrow_\$ \{0,1\}^* \\ (k_i)_{i\in[d]} \leftarrow_\$ \mathsf{PCG.Gen}(1^\lambda,(\rho_\sigma)_{\sigma\in T},(\tilde\rho_{\tilde\sigma})_{\tilde\sigma\in[d]\setminus T}) \end{array} \right\}$$

are computationally close.

We first observe that by changing the definition of the underlying multi-point function according to the programmed inputs, the previously constructed multiparty PCG for constant degree correlations based on multiparty DPFs in section 4.2 can be adapted to satisfy the programmability property defined above. We formally state this in Lemma 1 and explicitly show the construction in Protocol 5.

Lemma 1. *Setting* $N = \mathsf{tpp} = d$, *Protocol 2 can be adapted to a* d*-party programmable PCG for degree-*d *correlation* $\mathcal{C}_d^n$, *as per Definition 10.*

Proof (Proof of Lemma 1). To allow programmability, we slightly tweak the construction in Protocol 2 and describe the construction in Protocol 5. Intuitively, we change the definition of the multi-point function in $\mathsf{PCG.Gen}$ to accord with the programmed inputs.

Protocol 5: Programmable d-party PCG for degree-d Correlation $\mathcal{C}_d^n$ from d-party DPF

Parameters: $1^\lambda, n, n', t, p, d \in \mathbb{N}$, where $n' > n$. Let $\mathbf{C}$ be a code generation algorithm such that $H_{n',n} \leftarrow\$ \mathbf{C}(n',n,\mathbb{F}_p)$. Let $\mathsf{PRG} : \{0,1\}^\lambda \to \mathbb{F}_p^{n^d}$ be a PRG.

$\mathsf{PCG.Gen}(1^\lambda,\rho_1,\ldots,\rho_d)$:

1. For each $\sigma\in[d]$, interpret ρ_σ as a t-sparse vector $\mathbf{e}_\sigma \in \mathbb{F}_p^{n'}$.
2. Let $f : [(n')^d] \to \mathbb{F}_p$ be the multi-point function with t^d points, where $f(i)$ returns the i^{th} coordinate of $\bigotimes_{j=1}^d \mathbf{e}_j$.
3. Compute $(K_1^{\mathsf{fss}}, K_2^{\mathsf{fss}},\ldots,K_d^{\mathsf{fss}}) \leftarrow\$ \mathsf{MPFSS}_d.\mathsf{Gen}(1^\lambda, f)$.[a]
4. For every $i,j\in[d]$ with $i<j$, sample PRG seeds $s^{ij} \leftarrow\$ \{0,1\}^\lambda$.
5. For each $\sigma\in[d]$, let $k_\sigma \leftarrow (n,\rho_\sigma,K_\sigma^{\mathsf{fss}},\{s^{j\sigma}\}_{1\le j<\sigma},\{s^{\sigma j}\}_{\sigma<j\le d})$.
6. Output $(k_\sigma)_{\sigma\in[d]}$.

$\mathsf{PCG.Expand}(\sigma,k_\sigma)$:

1. On input (σ,k_σ), parse k_σ as $(n,\rho_\sigma,K_\sigma^{\mathsf{fss}},\{s^{j\sigma}\}_{1\le j<\sigma},\{s^{\sigma j}\}_{\sigma<j\le d})$ and interpret ρ_σ as $\mathbf{e}_\sigma$.

2. For evert $j \neq \sigma$, compute $t_{\sigma j} = \mathsf{PRG}(s^{\sigma j})$ if $\sigma < j$ and $t_{\sigma j} = \mathsf{PRG}(s^{j\sigma})$ otherwise.
3. Compute $\mathbf{v}_\sigma \leftarrow \mathsf{MPFSS}_d.\mathsf{FullEval}(\sigma, K_\sigma^{\mathsf{fss}})$. \\ Note that $\mathbf{v}_\sigma \in \mathbb{F}_p^{(n')^d}$.
4. Compute $\mathbf{R}_\sigma \leftarrow H_{n',n} \cdot \mathbf{e}_\sigma$ and $\mathbf{S}_\sigma \leftarrow (H_{n',n})^{\otimes d} \cdot \mathbf{v}_\sigma + \sum_{1 \le j < \sigma} t_{j\sigma} - \sum_{\sigma < j \le d} t_{j\sigma}$. \\ Note that $\mathbf{S}_\sigma \in (\mathbb{F}_p^n)^d$ and $\sum_{\sigma \in [d]} \mathbf{S}_\sigma = \bigotimes_{\sigma=1}^d \mathbf{R}_\sigma$.
5. Output $(\mathbf{R}_\sigma, \mathbf{S}_\sigma)$.

[a] MPFSS_d is an instance of distributed multi-point function with domain size $(n')^d$.

According to our construction, for each $\phi_\sigma : \{0,1\}^* \to \mathbb{F}_p^n$ and $\rho \in \{0,1\}^*$, $\phi_\sigma(\rho)$ first interprets ρ as a t-sparse vector $\mathbf{e} \in \mathbb{F}_p^{n'}$ and then computes $\mathbf{R} = H_{n',n} \cdot \mathbf{e}$. The correctness property follows from the correctness of the underlying MPFSS_d, the LPN assumption, and the security of PRG. The programmability holds due to our definition of $(\phi_\sigma)_{\sigma \in [d]}$. The programmable security stems out from the security of the underlying MPFSS_d which guarantees that for any set $T \subsetneq [d]$ of corrupted parties, the generated FSS keys of corrupted parties, $(K_\sigma^{\mathsf{fss}})_{\sigma \in T}$, do not leak any information of $(\rho_{\tilde{\sigma}})_{\tilde{\sigma} \in [d] \setminus T}$. We omit the formal proof as it is almost the same as that of Theorem 4 which constructs N-party PCG for tensor power correlations from N-party DPF.

Based on such a *programmable* D-party PCG for degree-D correlation, we then present a general transformation to derive an M-party PCG construction for degree-D correlation. Such an M-party PCG compresses the correlation $\mathcal{C}_{D,M}^N$, where each party P_σ takes the σ-th additive share of N random elements $r_1, r_2, \ldots, r_N$ together with all degree-D monomials in $r_1, r_2, \ldots, r_N$. Alternatively, with this M-party PCG, fixing an arbitrary N-variate polynomial $P(r_1, r_2, \ldots, r_N)$ of degree D, all parties are able to obtain their additive shares of $r_1, r_2, \ldots, r_N$ and $P(r_1, r_2, \ldots, r_N)$, which will be denoted by additive correlation $\mathcal{C}_{P,M}$. It is clear that $\mathcal{C}_{P,M}$ is a reverse-sampleable correlation.

At a high-level, letting $\mathbf{r} = (r_1, \ldots, r_N)$, the transformation works by invoking D-party PCG for degree-D correlation for every D-tuple $\tau = (j_1, \ldots, j_D) \in [M]^D$ to generate short seeds which expand to additive shares of $\bigotimes_{\ell=1}^D \mathbf{r}_{j_\ell}$, where $(\mathbf{r}_\sigma)_{\sigma \in [M]}$ forms an additive share of $\mathbf{r}$. We emphasize that, while D is restricted to be a constant because the length of the seeds generated by D-party PCG will explode as the degree increases, M can be polynomials in security parameter λ, which benefits from our construction based on programmable PCGs and the programmability is used to maintain the consistency across different monomials. We show such a transformation in Protocol 6 and summarize its functionality and computation complexity in Lemma 2. For each D-tuple $\tau \in [M]^D$, we abuse the notation as $\sigma \in \tau$ if there is a component in τ equal to σ.

Protocol 6: **M-party PCG from D-party PCG**

Parameters: Let $\mathsf{PCG}_D = (\mathsf{PCG}_D.\mathsf{Gen}, \mathsf{PCG}_D.\mathsf{Expand})$ be a programmable D-party PCG for degree-D correlation $\mathcal{C}_D^N$ with each key of size $s_D(\lambda)$. Let $P : \mathbb{G}^N \to \mathbb{G}$ be an N-variable degree-D polynomial defined as $(r_1, r_2, \ldots, r_N) \mapsto \sum_{\tau_1 = (i_1, i_2, \ldots, i_D) \in [N]^D} c_{\tau_1} \cdot r_{i_1} r_{i_2} \cdots r_{i_D}$. Here we treat P as a linear combination of degree-D monomials in N variables for simplicity.
$\mathsf{PCG}_M.\mathsf{Gen}(1^\lambda)$:

1. For each $\sigma \in [M]$, sample random $\rho_\sigma \leftarrow\$ \{0,1\}^*$ as specified by programmability property.
2. For every D-tuple $(j_1, j_2, \ldots, j_D) \in [M]^D$ denoted by τ_2, run $(k_{\tau_2}^{j_1}, k_{\tau_2}^{j_2}, \ldots, k_{\tau_2}^{j_D}) \leftarrow\$ \mathsf{PCG}_D.\mathsf{Gen}(1^\lambda, \rho_{j_1}, \rho_{j_2}, \ldots, \rho_{j_D})$.
3. For every $i, j \in [M]$ with $i < j$, sample PRG seed $s^{ij} \leftarrow_\$ \{0,1\}^\lambda$.
4. For each $\sigma \in [M]$, output $k_\sigma = (\{s^{j\sigma}\}_{1 \le j < \sigma}, \{s^{\sigma j}\}_{\sigma < j \le M}, \{k_{\tau_2}^\sigma\}_{\{\tau_2 \in [M]^D \mid \sigma \in \tau_2\}})$.

$\mathsf{PCG}_M.\mathsf{Expand}(\sigma, k_\sigma)$:

1. Parse k_σ as $(\{s^{j\sigma}\}_{1 \le j < \sigma}, \{s^{\sigma j}\}_{\sigma < j \le M}, \{k_{\tau_2}^\sigma\}_{\{\tau_2 \in [M]^D \mid \sigma \in \tau_2\}})$.
2. For every $j \neq \sigma$, compute $t_{\sigma j} = \mathsf{PRG}(s_{\sigma j})$ if $\sigma < j$ and $t_{j\sigma} = \mathsf{PRG}(s_{j\sigma})$ otherwise.

3. For D-tuple $(j_1, j_2, \ldots, j_D) \in [M]^D$ denoted by τ_2 such that $j_{\ell_{\tau_2}(\sigma)} = \sigma$ for some $\ell_{\tau_2}(\sigma) \in [D]$, compute $(\mathbf{R}_{\tau_2,\sigma}, \mathbf{S}_{\tau_2,\sigma}) \leftarrow \mathsf{PCG}_D.\mathsf{Expand}(\ell_{\tau_2}(\sigma), k^{\sigma}_{\tau_2})$. Notice that by programmability, $\mathbf{R}_{\tau_2,\sigma} = \phi(\rho_\sigma) \triangleq \mathbf{r}_\sigma \in \mathbb{F}_p^N$. Let $\mathbf{S}_{\tau_2,\sigma} = (S_{\tau_2,\sigma,\tau_1})_{\tau_1 \in [N]^D}$.
4. Output $\mathbf{r}_\sigma$ and $z_\sigma = \sum_{\tau_1 \in [N]^D} c_{\tau_1} \cdot \sum_{\{\tau_2 \in [M]^D \mid \sigma \in \tau_2\}} S_{\tau_2,\sigma,\tau_1} + \sum_{1 \le j < \sigma} t_{j\sigma} - \sum_{\sigma < j \le M} t_{\sigma j}$.

Lemma 2. *Let $\mathsf{PCG}_D = (\mathsf{PCG}_D.\mathsf{Gen}, \mathsf{PCG}_D.\mathsf{Expand})$ be a programmable PCG for degree-D correlation $\mathcal{C}_D^N$ with each key of size $s_D(\lambda)$. Then the construction $\mathsf{PCG}_M = (\mathsf{PCG}_M.\mathsf{Gen}, \mathsf{PCG}_M.\mathsf{Expand})$ in Protocol 6 is an M-party PCG for additive degree-D correlation $\mathcal{C}_{P,M}$ (specified by an N-variable polynomial $P : \mathbb{G}^N \to \mathbb{G}$ which is a linear combination of degree-D monomials in N variables) with the following properties.*

- $\mathsf{PCG}_M.\mathsf{Gen}(1^\lambda)$ *runs M^D executions of* $\mathsf{PCG}_D.\mathsf{Gen}$*, outputs key $(k_\sigma)_{\sigma \in [M]}$, each k_σ has size $O(M^{D-1}) \cdot s_D(\lambda) + (M-1) \cdot \lambda$ bits.*
- $\mathsf{PCG}_M.\mathsf{Expand}(\sigma, k_\sigma)$ *runs $O(M^{D-1})$ executions of* $\mathsf{PCG}_D.\mathsf{Expand}$ *and makes $(M-1)$ evaluations of a pseudorandom generator.*

Proof (Proof of Lemma 2).

Correctness. We first focus on the correctness. By our construction, each ρ_j is sampled uniformly at random. Since $\mathbf{r} = \sum_{j \in [M]} \mathbf{r}_j = \sum_{j \in [M]} \phi(\rho_j)$, then $\mathbf{r} = (r_i)_{i \in [N]}$ is random and $(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_M)$ forms a random additive sharing of $\mathbf{r}$ by the correctness and programmability of PCG_D. Moving to the distribution of $(z_\sigma)_{\sigma \in [M]}$, by correctness and programmability of PCG_D, we know that for every D-tuple $(i_1, i_2, \ldots, i_D) \in [N]^D$ denoted by τ_1 and D-tuple $(j_1, j_2, \ldots, j_D) \in [M]^D$ denoted by τ_2, the following holds except with negligible probability

$$\sum_{\ell=1}^{D} \mathbf{S}_{\tau_2, j_\ell} = \bigotimes_{\ell=1}^{D} R_{\tau_2, j_\ell} = \bigotimes_{\ell=1}^{D} \mathbf{r}_{j_\ell},$$

where we have $(k^{j_\ell}_{\tau_2})_{\ell \in [D]} \leftarrow \mathsf{PCG}_D.\mathsf{Gen}(1^\lambda, \rho_{j_1}, \ldots, \rho_{j_D})$ and $(\mathbf{R}_{\tau_2,j_\ell}, \mathbf{S}_{\tau_2,j_\ell}) \leftarrow \mathsf{PCG}_D.\mathsf{Expand}(j_\ell, k^{j_\ell}_{\tau_2})$ for each $\ell \in [D]$, which means with overwhelming probability,

$$\begin{aligned}
\sum_{\sigma=1}^{M} z_\sigma &= \sum_{\sigma=1}^{M} \Big(\sum_{\tau_1 \in [N]^D} c_{\tau_1} \cdot \sum_{\{\tau_2 \in [M]^D \mid \sigma \in \tau_2\}} S_{\tau_2,\sigma,\tau_1} + \sum_{1 \le j < \sigma} t_{j\sigma} - \sum_{\sigma < j \le M} t_{\sigma j} \Big) \\
&= \sum_{\tau_1 \in [N]^D} c_{\tau_1} \cdot \sum_{\sigma=1}^{M} \sum_{\{\tau_2 \in [M]^D \mid \sigma \in \tau_2\}} S_{\tau_2,\sigma,\tau_1} + \sum_{i<j} (t_{ij} - t_{ji}) \\
&= \sum_{\tau_1 \in [N]^D} c_{\tau_1} \cdot \sum_{\tau_2 \in [M]^D} \sum_{\ell=1}^{D} S_{\tau_2, j_\ell, \tau_1} \\
&= \sum_{\tau_1 \in [N]^D} c_{\tau_1} \cdot \sum_{\tau_2 \in [M]^D} \Big(\bigotimes_{\ell=1}^{D} \mathbf{r}_{j_\ell}\Big)_{\tau_1} = \sum_{\tau_1 \in [N]^D} c_{\tau_1} \cdot \Big(\bigotimes^{D} \mathbf{r}\Big)_{\tau_1} \\
&= \sum_{\tau_1 \in [N]^D} c_{\tau_1} \cdot \prod_{i \in \tau_1} r_i = P(r_1, r_2, \ldots, r_N).
\end{aligned}$$

Moreover, from pairwise pseudorandom offsets t_{ij} which are independent of $\{\rho_j\}_{j \in [M]}$, $(\sum_{1 \le j < \sigma} t_{j\sigma} - \sum_{\sigma < j \le M} t_{\sigma j})_{\sigma \in [M]}$ forms a random additive sharing of 0 and thus $(z_\sigma)_{\sigma \in [M]}$ forms a random additive sharing of $P(r_1, r_2, \ldots, r_N)$, which completes the correctness.

Security. Now we focus on the security. Let $T \subsetneq [M]$ denote the set of corrupted parties. By definition of PCG security, we aim to prove given corrupted parties' keys $(k_\sigma)_{\sigma \in T}$, the distribution of expanded honest parties' shares, $(\mathbf{r}_{\tilde{\sigma}}, z_{\tilde{\sigma}})_{\tilde{\sigma} \in [M] \setminus T}$, using honest parties' keys is indistinguishable from that generated by using reverse sampling algorithm, conditioned on expanded corrupted parties' shares, $(\mathbf{r}_\sigma, z_\sigma)_{\sigma\in T}$. Formally, the goal is to prove the following two distributions,

$$\mathcal{D}^{\mathrm{real}} \triangleq \left\{ (k_\sigma)_{\sigma\in T}, (\mathbf{r}_{\tilde\sigma}, z_{\tilde\sigma})_{\tilde\sigma\in[M]\setminus T} \middle| \begin{array}{l} (k_1, k_2, \ldots, k_M) \leftarrow \mathsf{PCG}_M.\mathsf{Gen}(1^\lambda) \\ (\mathbf{r}_{\tilde\sigma}, z_{\tilde\sigma}) \leftarrow \mathsf{PCG}_M.\mathsf{Expand}(\tilde\sigma, k_{\tilde\sigma}) \forall \tilde\sigma \in [M]\setminus T \end{array} \right\}$$

and

$$\mathcal{D}^{\mathrm{sim}} \triangleq \left\{ (k_\sigma)_{\sigma\in T}, (\mathbf{r}_{\tilde\sigma}, z_{\tilde\sigma})_{\tilde\sigma\in[M]\setminus T} \middle| \begin{array}{l} (k_1, k_2, \ldots, k_M) \leftarrow \mathsf{PCG}_M.\mathsf{Gen}(1^\lambda) \\ (\mathbf{r}_\sigma, z_\sigma) \leftarrow \mathsf{PCG}_M.\mathsf{Expand}(\sigma, k_\sigma) \forall \sigma \in T \\ (\mathbf{r}_{\tilde\sigma}, z_{\tilde\sigma})_{\tilde\sigma\in[M]\setminus T} \leftarrow \mathsf{RSample}(T, (\mathbf{r}_\sigma, z_\sigma)_{\sigma\in T}) \end{array} \right\},$$

are computationally indistinguishable. We first observe that due to the pairwise secret pseudorandom offsets $t_{i'j'} = \mathsf{PRG}(s_{i'j'})$, given $(k_\sigma)_{\sigma\in T}$ and $(\mathbf{r}_{\tilde\sigma})_{\tilde\sigma\in[M]\setminus T}$, the joint distribution of $(z_{\tilde\sigma})_{\tilde\sigma\in[M]\setminus T}$ is indistinguishable from random, up to the preserved sum $\sum_{\tilde\sigma\in[M]\setminus T} z_{\tilde\sigma}$ as required.

It then remains to show that given $(k_\sigma)_{\sigma\in T}$, the expanded honest values $(\mathbf{r}_{\tilde\sigma})_{\tilde\sigma\in[M]\setminus T}$ are pseudorandom. It is sufficient to consider an extreme case, where all but one party $\overline{\sigma} \in [M]$ is corrupted. By the correctness and programmability of the underlying PCG_D, it suffices to prove the following two distributions,

$$\mathcal{D}^{\mathrm{real}'} \triangleq \left\{ \left\{ k^\sigma_{\tau_2} \middle| \begin{array}{l} \tau_2 \in [M]^D, \\ \sigma \in \tau_2, \sigma \neq \overline{\sigma} \end{array} \right\}, \phi(\rho_{\overline{\sigma}}) \middle| \begin{array}{l} \rho_{\overline{\sigma}} \leftarrow \{0,1\}^*, \\ (\rho_\sigma)_{\sigma\neq\overline{\sigma}} \leftarrow \{0,1\}^*, \\ \left\{ k^\sigma_{\tau_2} \middle| \begin{array}{l} \tau_2 \in [M]^D, \\ \sigma \in \tau_2 \end{array} \right\} \leftarrow \mathsf{Seeds}(\rho_{\overline{\sigma}}, (\rho_\sigma)_{\sigma\neq\overline{\sigma}}) \end{array} \right\}$$

and

$$\mathcal{D}^{\mathrm{sim}'} \triangleq \left\{ \left\{ k^\sigma_{\tau_2} \middle| \begin{array}{l} \tau_2 \in [M]^D, \\ \sigma \in \tau_2, \sigma \neq \overline{\sigma} \end{array} \right\}, \phi(\rho_{\overline{\sigma}}) \middle| \begin{array}{l} \rho_{\overline{\sigma}} \leftarrow \{0,1\}^*, \\ (\rho_\sigma)_{\sigma\neq\overline{\sigma}} \leftarrow \{0,1\}^*, \\ \left\{ k^\sigma_{\tau_2} \middle| \begin{array}{l} \tau_2 \in [M]^D, \\ \sigma \in \tau_2 \end{array} \right\} \leftarrow \mathsf{Seeds}_{\overline{\sigma}}((\rho_\sigma)_{\sigma\neq\overline{\sigma}}) \end{array} \right\},$$

are computationally indistinguishable, where

- Seeds is an efficiently computable algorithm that runs Step 2 of $\mathsf{PCG}_M.\mathsf{Gen}$ with programmed inputs $(\rho_{\overline{\sigma}}, (\rho_\sigma)_{\sigma\neq\overline{\sigma}})$ fed to it and outputs all the seeds generated by invoking $\mathsf{PCG}_D.\mathsf{Gen}$.
- $\mathsf{Seeds}_{\overline{\sigma}}$ is an efficiently computable algorithm that runs Step 2 of $\mathsf{PCG}_M.\mathsf{Gen}$ with programmed inputs $(\rho_{i,\sigma})_{i\in[N],\sigma\neq\overline{\sigma}}$ fed to it except for each invocation of $\mathsf{PCG}_D.\mathsf{Gen}$ labeled with $\tau_2 = (j_1, \ldots, j_D)$ such that $j_\ell = \overline{\sigma}$ for some $\ell \in [D]$, Seeds' uses a fresh randomness $\tilde\rho_{\tau_2,\overline{\sigma}}$ instead of using the corresponding $\rho_{\overline{\sigma}}$ to generate the keys. After running all the invocations of $\mathsf{PCG}_D.\mathsf{Gen}$, $\mathsf{Seeds}_{\overline{\sigma}}$ outputs all the seeds generated.

We will prove this using a sequence of hybrid arguments, where we may replace each invocation of $\mathsf{PCG}_D.\mathsf{Gen}$ using honest $r_{i,\tilde\sigma}$ with fresh randomness one at a time. Intuitively, we will reduce such an indistinguishability to the programmable security of PCG_D though the reduction will lose a factor of $O(M^{D-1})$ in advantage. In particular, for the k-th hybrid, the hybrid distribution $\mathcal{D}^k$ is defiend as follows.

- Pick randomness $\rho_{\overline{\sigma}}, (\rho_\sigma)_{\sigma\neq\overline{\sigma}}$.
- Run Seeds with programmed inputs $(\rho_\sigma, (\rho_\sigma)_{\sigma\neq\overline{\sigma}})$ except that, for the first k invocations of $\mathsf{PCG}_D.\mathsf{Gen}$ labeled with $\tau_2 = (j_1, \ldots, j_D)$ such that $j_\ell = \overline{\sigma}$ for some $\ell \in [D]$, $\mathsf{PCG}_D.\mathsf{Gen}$ will use a fresh randomness $\tilde\rho_{\tau_2,\overline{\sigma}}$ rather than $\rho_{\overline{\sigma}}$ to generate the seeds.
- Output $\{k^\sigma_{\tau_2} \mid \tau_2 \in [M]^D, \sigma \in \tau_2, \sigma \neq \overline{\sigma}\}, \phi(\rho_{\overline{\sigma}})$.

Given a distinguisher $\mathcal{A}$ for $\mathcal{D}^k$ and $\mathcal{D}^{k+1}$, we can construct a distinguisher $\mathcal{B}$ for the programmable security property of PCG_D, with the same advantage. In particular, supposing the $(k+1)$-th invocation of $\mathsf{PCG}_D.\mathsf{Gen}$ is labeled with $\tau'_2 = (j'_1, \ldots, j'_D)$ with $j'_\ell = \overline{\sigma}$ for some $\ell \in [D]$, $\mathcal{B}$ receives $(k^\sigma_{\tau'_2})_{\sigma\in\tau'_2,\sigma\neq\overline{\sigma}}$ and $(\rho_{j'_\ell})_{\ell\in[D]}$ from its experiment. Then $\mathcal{B}$ works as follows to simulate the view of $\mathcal{A}$.

- $\mathcal{B}$ additionally samples $(\rho_\sigma)_{\sigma\in[M],\sigma\notin\tau'_2}$.

– $\mathcal{B}$ runs Seeds with programmed inputs $(\rho_\sigma)_{\sigma\neq\overline{\sigma}}$ except that, for the first k invocations of $\mathsf{PCG}_D.\mathsf{Gen}$ labeled with τ_2 such that $j_\ell = \overline{\sigma}$ for some $\ell \in [D]$, $\mathsf{PCG}_D.\mathsf{Gen}$ will use a fresh randomness $\tilde{\rho}_{\tau_2,\overline{\sigma}}$ rather than $\rho_{\overline{\sigma}}$ to generate the seeds and $\mathcal{B}$ **does not** run the invocation of $\mathsf{PCG}_D.\mathsf{Gen}$ labeled with τ_2'.
– $\mathcal{B}$ obtains $\{k^\sigma_{\tau_2} \mid \tau_2 \in [M]^D, \sigma \in \tau_2 \text{ such that } \tau_2 \neq \tau_2', \sigma \neq \overline{\sigma}\}, \phi(\rho_{\overline{\sigma}})$ as well as the values of $(k^\sigma_{\tau_2'})_{\sigma\in\tau_2',\sigma\neq\overline{\sigma}}$ received from its experiment and forwards them to $\mathcal{A}$.

$\mathcal{B}$ forwards the reply of $\mathcal{A}$ to its own experiment. Notice that if $\mathcal{B}$ obtains the real $\rho_{\overline{\sigma}}$ from its own experiment, then the resulting distribution is identical to $\mathcal{D}^k$, whereas if $\mathcal{B}$ obtains a simulated $\rho_{\overline{\sigma}}$, $\mathcal{B}$ simulates $\mathcal{D}^{k+1}$. Therefore, $\mathcal{B}$ successfully breaks the programmable security of PCG_D, whenever $\mathcal{A}$ successfully distinguishes $\mathcal{D}^k$ and $\mathcal{D}^{k+1}$.

Since there are only $O(M^{D-1})$ invocations of $\mathsf{PCG}_D.\mathsf{Gen}$ labeled with $\tau_2 = (j_1, \dots, j_D)$ such that $j_\ell = \overline{\sigma}$ for some $\ell \in [D]$, the distributions of $\mathcal{D}^{\mathrm{real}'}$ and $\mathcal{D}^{\mathrm{sim}'}$ are computationally indistinguishable, where such an indistinguishability is reduced to the programmable security of PCG_D with $O(M^{D-1})$ as the reduction factor. By previous analysis, we prove the PCG security of PCG_M. ◻

5.4 Corollaries

We spell out some corollaries of our results of the previous section. First, combining lemma 2 (M-party PCG from d-party programmable PCG) with lemma 1 (d-party programmable PCG from d-party DPF) and theorem 7 (setting $m = \log_2(d) + 1$), we get:

Corollary 1. *Fix integers $m, d > 0$ and polynomials M, N. Assume the existence of a pseudorandom generator $G : \{0,1\}^\lambda \to \{0,1\}^{\lambda\cdot(\lambda+1)}$ that can be computed by a degree-c polynomial and define $d_{2^m,d} = (1/3)\cdot(3, c, d) \uparrow m$. Then assuming in addition the dual $\mathbb{F}\text{-}\mathsf{LPN}(\lambda^3/2+1, \lambda^3, \lambda^{1/(2d_{d,d})})$ assumption, there exists an M-party pseudorandom correlation generator $(\mathsf{PCG.Gen}, \mathsf{PCG.Expand})$ for any degree-d length-N tensor power correlation with keys of size $O(M^{d-1}\cdot\lambda^3)$, and where $\mathsf{PCG.Gen}$ can be computed using $6d_{d,d}\cdot O(M^d) = O(M^d)$ calls to G and $O(M^d\cdot\lambda^3)$ additional operations.*

Via theorem 5, this PCG can be converted to an M-party HSS for N-variate degree-d polynomials with shares of size $N + O(M^{d-1}\cdot\lambda^3)$:

Corollary 2. *Fix integers $m, d > 0$ and polynomials M, N. Under the same assumptions as corollary 1, there exists an M-party HSS HSS for degree-d N-variate correlations with keys of size $N + O(M^{d-1}\cdot\lambda^3)$ where $\mathsf{HSS.Share}$ can be computed using $O(M^d)$ calls to G and $O(N\cdot\mathsf{poly}(\lambda) + M^d\cdot\lambda^3)$ additional operations.*

Eventually, using theorem 6 to combine this HSS with the 2-party DPF of theorem 3, we get:

Corollary 3. *Fix integers $m, d > 0$ and a polynomial M. Under the same assumptions as corollary 1, there exists a $2M$-party distributed point function over a domain of size λ^d with keys of size $O(M^{c^d-1}\cdot\lambda^3)$.*

We note that the last corollary is meaningful for values of M significantly smaller than λ (since there always exists a trivial multiparty DPF with keys of size $O(\lambda^d)$: it suffices to share the truth table of the point function). In particular, for any $M = \mathsf{polylog}(\lambda)$, it yields an M-party DPF with key size $\tilde{O}(\lambda^3)$.

6 Applications

6.1 Private information retrieval

A 2-round M-server private information retrieval [CGKS95,KO97] involves M servers $S_1, \cdots, S_M$, each holding a copy of a string $z \in \{0,1\}^{\lambda^d}$ (the database) and a client C holding an integer $i \in [\lambda^d]$ (the query). The client C sends a single message to each server, and the servers send a single message back to the client (in particular, the servers do not communicate). Formally,

Definition 11 (2-round M-server PIR). *Fix a constant d. A 2-round M-server private information retrieval scheme for databases of size λ^d is a triple of algorithms* ($\mathsf{Query}, \mathsf{Answer}, \mathsf{Output}$) *with the following syntax:*

- $\mathsf{Query}(i)$: *on input $i \in [\lambda^d]$, output an M-tuple $(q_1, \cdots, q_M)$ of queries.*
- $\mathsf{Answer}(q, z)$: *on input q and a database $z \in \{0,1\}^{\lambda^d}$, output an answer a.*
- $\mathsf{Output}(a_1, \cdots, a_M)$: *on input an M-tuple $(a_1, \cdots, a_M)$, output a bit b.*

A PIR must be correct *and* private*:*

Correctness: *there exists a negligible function μ such that for every $\lambda \in \mathbb{N}$, $z \in \{0,1\}^{\lambda^d}$, $i \in [\lambda^d]$, and every $(q_1, \cdots, q_M)$ in the support of* $\mathsf{Query}(i)$*, denoting $a_j = \mathsf{Answer}(q_j, z)$ for $j = 1$ to M, it holds that*

$$\Pr[\mathsf{Output}(a_1, \cdots, a_M) = z_i] \geq 1 - \mu(\lambda).$$

Security: *there exists a negligible function μ such that for every PPT adversary $\mathcal{A}$, large enough $\lambda \in \mathbb{N}$, $j \in [M]$, $(i_0, i_1) \in [\lambda^d]^2$, and $z \in \{0,1\}^n$:*

$$|\Pr[\mathcal{A}((q_k^0)_{k\in[m]\setminus\{j\}}) = 1] - \Pr[\mathcal{A}((q_k^1)_{k\in[m]\setminus\{j\}}) = 1]| \leq \mu(\lambda, n),$$

where the probability is over the choice of $(q_1^b, \cdots, q_M^b) \leftarrow \mathsf{Query}(i_b)$ for $b = 0, 1$ and the random coins of $\mathcal{A}$.

The main measure of the efficiency of a PIR scheme is its communication complexity, which is the maximum number of bits exchanged between the user and the servers. We call *upload communication* the maximum size of any query q_j over any query i, server index j, and random coins for Query. We say that a scheme has *optimal download rate* and *additive reconstruction* if $|a_j| = 1$ for $j = 1$ to m (for all possible databases and queries) and $\mathsf{Output}(a_1, \cdots, a_M) = \bigoplus_{j=1}^M a_j$.

The study of private information retrieval has a long history. In our definition above, we focus on the setting of maximum corruption (up to $M - 1$ servers might collude). It is well known that an M-party DPF over a domain of size λ^d (and range $\{0,1\}$) immediately implies a 2-round M-party PIR with optimal download rate and additive reconstruction. The construction proceeds as follows:

- $\mathsf{Query}(i)$: define the point function $f = f_{i,1}$ that evaluates to 1 on i and output $(q_1, \cdots, q_M) \leftarrow \mathsf{DPF.Share}(f)$.
- $\mathsf{Answer}(q, z)$: parse q as a key for DPF. Set $v = (\mathsf{DPF.Eval}(q, x))_{x\in[\lambda^d]}$. Return $a = \langle v, z\rangle \bmod 2$ ($\langle\cdot\rangle$ denotes the inner product).

We refer the reader to [GI14] for the formal construction and security analysis. Plugging our construction of multiparty DPF from corollary 3, we get:

Corollary 4. *Fix an integers $d > 0$ and a polynomial M. Assume the existence of a pseudorandom generator $G : \{0,1\}^\lambda \to \{0,1\}^{\lambda\cdot(\lambda+1)}$ that can be computed by a degree-c polynomial and define $\Delta = (2/3) \cdot (3, c, d) \uparrow \log_2(d)$. Then assuming in addition the dual $\mathbb{F}$-$\mathsf{LPN}(\lambda^3/2 + 1, \lambda^3, \lambda^{1/\Delta})$ assumption, there exists a 2-round M-party private information retrieval scheme for databases of size λ^d with upload rate $O(M^{d-1} \cdot \lambda^3)$, optimal download rate, and additive reconstruction.*

As shown in [GI14], the construction of PIR from DPF can be extended to other settings, such as PIR by keywords (where the client wants to know whether a word w matches any entry from a database of λ^d words held by M client) and PIR by keywords with payloads (where the client receives a payload p_i if w matches the word w_i in the database). We omit the list of straightforward (similar) corollaries.

Remark 2. The LPN and MQ assumptions both require the parties to agree on random matrices. There are several ways to deal with this technicality:

- If the parties are given access to a common random string, they can agree on these matrices with no communication. In this case, the definition of PIR must be slightly modified to account for the use of a CRS, to let the adversary choose the database z after seeing the CRS.
- Alternatively, the parties can rely on the random oracle model to locally generate these matrices.
- Eventually, the parties can use *fixed* matrices, or matrices that can be sampled using a small number of random coins (e.g. by defining the matrices to be the output of a PRG). In this case, the security of the protocol relies on the LPN and MQ assumptions with respect to these specific matrices. We note that instantiating the matrices in LPN via a pseudorandom generator is a a very common assumption, made e.g. in most code-based signatures schemes [FJR22,CCJ23].

6.2 Secure computation with silent preprocessing

As an immediate consequence of our PCG for constant degree correlations, we also obtain the first (M-party) PCG for generating *authenticated Beaver triples*. Plugging this PCG in a maliciously-secure M-party protocol from authenticated Beaver triples (e.g. SPDZ [DPSZ12]) This implies a maliciously secure computation protocol with silent preprocessing, where the preprocessing communication scales as $O(M^3 \cdot \mathsf{poly}(\lambda))$, from the hardness of LPN and the existence of constant-degree PRGs. Previous works required either LWE [DHRW16], the DCR assumption on top of MQ and LPN [BCM23,CK24], or required ring-LPN and had a preprocessing communication scaling as $O(M^4 \cdot \mathsf{poly}(\lambda) \cdot \sqrt{s})$ for circuits of size s [AS22]. We omit the (straightforward) corollary.

6.3 M-party MPC with low communication

In this section, we show that our results give rise to an MPC protocol with communication smaller than the circuit size for all layered circuits. In contrast with the result of [DIJL23], that provides a dedicated construction of sublinear MPC from their sparse-LPN-based multiparty HSS, our result is obtained by simply invoking existing results that generically obtain sublinear MPC given a suitable PCG. This simplicity is a conceptual advantage over the approach of [DIJL23]. In [DIJL23], the authors had to rely on a dedicated approach because their multiparty HSS scheme has non-negligible correctness error, and as a consequence does *not* imply a PCG. We start by representing on $\mathcal{F}_C$ the ideal functionality for securely evaluating a layered arithmetic circuit C.

Functionality 1: $\mathcal{F}_C$

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

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.

Functionality 2: $\mathcal{F}_{\mathsf{Corr}}$

- **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 $\mathcal{A}$ 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 $\mathcal{A}$, 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 $\mathcal{A}$.
3. *Setup input masks:* On input a message $(\mathsf{setinputshare}, (\boldsymbol{r}_{\mathsf{in},\ell})_{\ell \in D})$ from $\mathcal{A}$ with $\forall \ell \in D, \boldsymbol{r}_{\mathsf{in},\ell} \in \mathbb{F}^n$, sample $(\boldsymbol{r}_{\mathsf{in},\ell})_{\ell \in H} \leftarrow (\mathbb{F}^n)^{|H|}$, 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 $\mathcal{A}$ with $\forall \ell \in D, \boldsymbol{r}_{i,\ell} \in \mathbb{F}^{s_i}$, sample $(\boldsymbol{r}_{i,\ell})_{\ell \in H} \leftarrow (\mathbb{F}^{s_i})^{|H|}$, 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:
      $$\boldsymbol{\pi}^{(i,j)} \leftarrow \left(1 \mid\mid \boldsymbol{r}_{\mathsf{in}}[U^{\mathsf{in}}_{i,j}] \mid\mid \boldsymbol{r}_i[U_{i,j}]\right)^{\otimes 2^k}.$$
      – Wait for a message $(\mathsf{setshare}, (i,j), (\boldsymbol{\pi}^{(i,j)}_\ell)_{\ell \in D})$ from $\mathcal{A}$ 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:
      $$\boldsymbol{\pi}^{(i,j)} \leftarrow \left(1 \mid\mid \boldsymbol{r}_{\mathsf{in}}[V^{\mathsf{in}}_{i,j}] \mid\mid \boldsymbol{r}_i[V_{i,j}]\right)^{\otimes 2^k}.$$
      – Wait for a message $(\mathsf{setoutputshare}, (i,j), (\boldsymbol{\pi}^{(i,j)}_\ell)_{\ell \in D})$ from $\mathcal{A}$ 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 $(\boldsymbol{r}_{\mathsf{in},\ell}, (\boldsymbol{r}_{i,\ell}, (\boldsymbol{\pi}^{(i,j)}_\ell)_{1 \le j \le \lceil s_{i+1}/\beta \rceil}, (\boldsymbol{\pi}^{(i,j)}_{\mathsf{out},\ell})_{1 \le j \le \lceil m_i/\beta \rceil})_{0 \le i < \lceil d/k \rceil})$ to each party P_ℓ.

Theorem 8 (Theorem 12 from [CM21]). *Let $k \le \log\log s - \log\log\log s$. There exists a protocol Π_C which (perfectly) securely implements the N-party functionality $\mathcal{F}_C$ in the $\mathcal{F}_{\mathsf{Corr}}$-hybrid model, against a static, passive, nonaborting adversary corrupting at most $N-1$ out of N parties, with communication complexity upper bounded by $N \cdot \left(O(n+m) + \frac{s}{k}\right) \cdot \log|\mathbb{F}|$.*

represent in $\mathcal{F}_{\mathsf{Corr}}$ the ideal *corruptible*[4] functionality for distributing (function-dependent) correlated randomness between the parties. The ideal functionality $\mathcal{F}_{\mathsf{Corr}}$ (which is reproduced verbatim from [CM21], with the authorization of the authors) samples a complex *subset tensor powers* correlation, where the parties receive shares of the tensor powers of strings derived from a pseudorandom $\boldsymbol{r}$, where the strings are of the form $(1||\boldsymbol{r}_S)$ for size-2^k subsets S determined by the topology of the circuit. It is a straightforward observation that a PCG for the 2^k-th tensor power correlation immediately implies a PCG for the (much more complex and circuit-dependent) subset tensor power correlation defined in $\mathcal{F}_{\mathsf{Corr}}$, simply because $(1||\boldsymbol{r})^{\otimes 2^k}$ contains *all* size-2^k products of entries of $(1||\boldsymbol{r})$, hence the substring tensor power correlation outputs effectively a subset of the entries of $(1||\boldsymbol{r})^{\otimes 2^k}$.[5]

Then, [BCG+19, Theorem 19] proves that given an ideal functionality that distributes the seeds of a PCG for a reverse-samplable correlation, the simple (no communication) protocol that lets the parties locally stretch their seeds into a pseudorandom correlation securely instantiates (in the malicious setting) the ideal corruptible functionality that distributes the corresponding correlation.

It remains to securely instantiate the functionality that distributes outputs of $\mathsf{PCG.Gen}$, where PCG is the construction from corollary 1. Using generic secure computation (e.g. GMW) with communication

[4] A functionality that distributes additive shares is corruptible if it lets the adversary define the outputs of the corrupted parties, and samples the honest parties' outputs to be consistent with the corrupted parties' outputs.

[5] The point of considering this more convoluted correlation in [CM21] is that when $k = \omega(1)$, storing this correlation does not require storing a superpolynomial amount of correlated randomness, while for a circuit with s gates, the PCG's output for the tensor power correlation grows as s^{2^k}. However, for the case of $k = O(1)$ which we consider here, using the tensor power correlation directly suffices.

and computation $O(M \cdot s \cdot \mathsf{poly}(\lambda))$ for M-party secure computation of a circuit of size s, M parties can securely compute the $O(M^d)$ invocations of G and $O(M^d \cdot \lambda^3)$ additional operations using $\mathsf{poly}(\lambda) \cdot M^d$ bits of communication. Combining Theorem 19 from [BCG+19] with theorem 8 and our PCG (corollary 1) with the (generic) approach to distributively generate PCG keys sketched above, we get:

Theorem 9. *There exists a universal constant θ such that for any constant k, there exists a protocol Π_C which securely implements the M-party functionality $\mathcal{F}_C$ in the standard model against a static, passive adversary corrupting at most $M-1$ out of M parties, with communication complexity upper bounded by*

$$M \cdot \left(O(n+m) + \frac{s}{k}\right) \cdot \log|\mathbb{F}| + \mathsf{poly}(\lambda) \cdot M^{\theta \cdot k}.$$

Remark 3. The protocol obtained in theorem 9 is in the *preprocessing model*: after a one-time preprocessing phase (independent of the inputs) with total communication $\mathsf{poly}(\lambda) \cdot M^{\theta \cdot k}$, the parties can securely compute an arbitrary layered arithmetic circuit C over $\mathbb{F}$ using total communication $\left(N \cdot O(n+m) + \frac{s}{k}\right) \cdot \log|\mathbb{F}|$. Furthermore, the online phase is *non-cryptographic*: it does not require any cryptographic primitives. This distinguishes this protocol from previous sublinear multiparty computation protocols based on FHE that require the parties to use cryptographic computations after the inputs have been defined.

6.4 *N*-party MPC with Sublinear Communication

The protocol outlined in the previous section falls short of providing truly sublinear communication: for any constant k, it communicates s/k field elements per parties. In this section, we sketch how to extend our results to obtain an N-party MPC protocol with total communication *sublinear* in the size of the circuit where N is any constant number of parties. Our protocol follows the technique of [CM21] to instantiate a PCG for the *subset tensor powers* correlation using the N-party HSS scheme outlined in Subsection 5.1.

We provide a sketch of our argument below. A formal proof of the theorem follows from a straightforward parametrization process which nevertheless involves importing a very large number of lemmas from [CM21], and hence it has been omitted. A complete treatment of this proof can be found in the full version of this work.

We now sketch how to obtain a low-communication, single-circuit N-party PCG for the stp correlation, which is sufficient to obtain an N-party protocol for evaluating arbitrary layered circuits with sublinear computation. The stp correlation is parametrized by 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 tpp and generates shares of $(\mathbf{r}, ((1_{\mathbb{F}} \| \mathbf{r}[S_i])^{\otimes \mathsf{tpp}})_{1 \le i \le \mathsf{n_s}})$, where $\mathbf{r} \in \mathbb{F}^w$ is random. We provide a formal definition of the ideal functionality that generates this correlation on $\mathcal{F}_{\mathsf{Corr}}$. Our PCG is instantiated by the generic compiler of [CM21], which takes as input any N-party DPF with domain size λ^d and key size $\mathsf{poly}(\lambda)$ and outputs a PCG $\mathsf{PCG_{stp}}$ via the transformation in Fig. 7 of [CM21]. A series of theorems starting from Theorem 15 to Theorem 20 of [CM21] then culminates in a proof that under the superpolynomial variant of LPN such a PCG is sufficient to obtain sublinear MPC for layered circuits. We now consider the application of this compiler to our DPF presented in Theorem 7. The bottleneck of the approach is the communication and computation of the resultant MPFSS scheme formed by adding together several instances of the DPF.

- **Communication:** The key size of our 2^m-DPF is $O(\lambda^3) = \mathsf{poly}(\lambda)$. This is sufficient for the purpose of the compiler.
- **Computation:** The evaluation algorithm of our 2^m-DPF can be computed by a degree-$d_{2^m,d}$ polynomial where $d_{2^m,d} = (1/3) \cdot (3^{1/d}c, d) \uparrow m$. This is polynomial for constant m. However, Section 6 of [CM21] shows that when the compiler is instantiated by the DPF of Theorem 7, the resulting total computation involves a term of the form $(\log s)^{d_{2^m,k}}$ where k is the sublinearity factor, which needs to be in $\omega(1)$. Since we require $(\log s)^{d_{2^m,k}} \le s^{O(1)}$, it follows that the exact upper bound for the sublinearity factor is dependent on 2^m, ie. the number of parties. If we let $\mathsf{itlog}(k, x)$ represent the k^{th} *iterated logarithm* of x, that is

$$\mathsf{itlog}(k, x) = \underbrace{\log(\log(\ldots(\log(x))\ldots))}_{k \text{ times}},$$

we get that for 2^m parties, $k \leq \mathsf{itlog}(m+1, s) - \mathsf{itlog}(m+2, s) + O(1)$. To remove dependence on m, we note that $\log^*(x) = o(\mathsf{itlog}(n, x))$ for any constant value of n. Hence, we can set $k = O(\log^*(s))$ or any arbitrarily slow-growing function in $o(\mathsf{itlog}(n, x))$, such as the inverse Ackermann function.

– **Assumption**: As in [CM21], setting parameters requires a superpolynomial flavor of the LPN assumption formed by setting $\lambda = o(\lambda')$ for some suitable λ'. However, instantiating the low-degree PRG with any compatible assumption —MQ, in our case— requires $\lambda_{\mathsf{MQ}} = \lambda_{\mathsf{LPN}}$, and hence we require a superpolynomial flavor of the MQ assumption as well.

We conclude with the following corollary of Theorem 7.

Theorem 10. *Let C be a layered arithmetic circuit of size s with n inputs and m outputs. For any constant N, there exists a protocol Π'_C which securely implements the N-party functionality $\mathcal{F}_C$ in the standard model against a static, passive adversary corrupting at most $N-1$ out of N parties, with communication complexity upper bounded by*

$$\left(O(n+m) + \frac{s}{\log^* s}\right) \cdot \log|\mathbb{F}| + \mathsf{poly}(\lambda).$$

References

ADOS22. Damiano Abram, Ivan Damgård, Claudio Orlandi, and Peter Scholl. An algebraic framework for silent preprocessing with trustless setup and active security. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 421–452. Springer, Cham, August 2022.

ARS24. Damiano Abram, Lawrence Roy, and Peter Scholl. Succinct homomorphic secret sharing. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part VI*, volume 14656 of *LNCS*, pages 301–330. Springer, Cham, May 2024.

AS22. Damiano Abram and Peter Scholl. Low-communication multiparty triple generation for SPDZ from ring-LPN. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *PKC 2022, Part I*, volume 13177 of *LNCS*, pages 221–251. Springer, Cham, March 2022.

BCG+17. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, and Michele Orrù. Homomorphic secret sharing: Optimizations and applications. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 2105–2122. ACM Press, October / November 2017.

BCG+19. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 489–518. Springer, Cham, August 2019.

BCG+20. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators from ring-LPN. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 387–416. Springer, Cham, August 2020.

BCM23. Elette Boyle, Geoffroy Couteau, and Pierre Meyer. Sublinear-communication secure multiparty computation does not require FHE. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part II*, volume 14005 of *LNCS*, pages 159–189. Springer, Cham, April 2023.

BDIR18. Fabrice Benhamouda, Akshay Degwekar, Yuval Ishai, and Tal Rabin. On the local leakage resilience of linear secret sharing schemes. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 531–561. Springer, Cham, August 2018.

BGI15. Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 337–367. Springer, Berlin, Heidelberg, April 2015.

BGI16a. Elette Boyle, Niv Gilboa, and Yuval Ishai. Breaking the circuit size barrier for secure computation under DDH. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 509–539. Springer, Berlin, Heidelberg, August 2016.

BGI16b. Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 1292–1303. ACM Press, October 2016.

BGI17. Elette Boyle, Niv Gilboa, and Yuval Ishai. Group-based secure computation: Optimizing rounds, communication, and computation. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 163–193. Springer, Cham, April / May 2017.

BGP06. Côme Berbain, Henri Gilbert, and Jacques Patarin. QUAD: A practical stream cipher with provable security. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 109–128. Springer, Berlin, Heidelberg, May / June 2006.

BKS19. Elette Boyle, Lisa Kohl, and Peter Scholl. Homomorphic secret sharing from lattices without FHE. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 3–33. Springer, Cham, May 2019.

CCJ23. Eliana Carozza, Geoffroy Couteau, and Antoine Joux. Short signatures from regular syndrome decoding in the head. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 532–563. Springer, Cham, April 2023.

CGKS95. Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *36th FOCS*, pages 41–50. IEEE Computer Society Press, October 1995.

CK24. Geoffroy Couteau and Naman Kumar. 10-party sublinear secure computation from standard assumptions. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part IX*, volume 14928 of *LNCS*, pages 39–73. Springer, Cham, August 2024.

CM21. Geoffroy Couteau and Pierre Meyer. Breaking the circuit size barrier for secure computation under quasi-polynomial LPN. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part II*, volume 12697 of *LNCS*, pages 842–870. Springer, Cham, October 2021.

COS+22. Ilaria Chillotti, Emmanuela Orsini, Peter Scholl, Nigel P. Smart, and Barry van Leeuwen. Scooby: Improved multi-party homomorphic secret sharing based on FHE. In Clemente Galdi and Stanislaw Jarecki, editors, *SCN 22*, volume 13409 of *LNCS*, pages 540–563. Springer, Cham, September 2022.

DHRW16. Yevgeniy Dodis, Shai Halevi, Ron D. Rothblum, and Daniel Wichs. Spooky encryption and its applications. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 93–122. Springer, Berlin, Heidelberg, August 2016.

DIJL23. Quang Dao, Yuval Ishai, Aayush Jain, and Huijia Lin. Multi-party homomorphic secret sharing and sublinear MPC from sparse LPN. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part II*, volume 14082 of *LNCS*, pages 315–348. Springer, Cham, August 2023.

DPSZ12. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, Berlin, Heidelberg, August 2012.

FJR22. Thibauld Feneuil, Antoine Joux, and Matthieu Rivain. Syndrome decoding in the head: Shorter signatures from zero-knowledge proofs. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 541–572. Springer, Cham, August 2022.

GI14. Niv Gilboa and Yuval Ishai. Distributed point functions and their applications. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 640–658. Springer, Berlin, Heidelberg, May 2014.

JKPT12. Abhishek Jain, Stephan Krenn, Krzysztof Pietrzak, and Aris Tentes. Commitments and efficient zero-knowledge proofs from learning parity with noise. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 663–680. Springer, Berlin, Heidelberg, December 2012.

KO97. Eyal Kushilevitz and Rafail Ostrovsky. Replication is NOT needed: SINGLE database, computationally-private information retrieval. In *38th FOCS*, pages 364–373. IEEE Computer Society Press, October 1997.

OSY21. Claudio Orlandi, Peter Scholl, and Sophia Yakoubov. The rise of paillier: Homomorphic secret sharing and public-key silent OT. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 678–708. Springer, Cham, October 2021.

RS21. Lawrence Roy and Jaspal Singh. Large message homomorphic secret sharing from DCR and applications. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part III*, volume 12827 of *LNCS*, pages 687–717, Virtual Event, August 2021. Springer, Cham.