On Compressing Linearly Shared Correlations

Geoffroy Couteau[1], Alexander Koch[2], Nikolas Melissaris[1], Peter Scholl[3], Sacha Servan-Schreiber[4], and Xiaxi Ye[5]

[1] CNRS, IRIF & Université Paris Cité
[2] Secorvo Security Consulting GmbH
[3] Aarhus University
[4] Tinfoil
[5] Tsinghua University

Abstract. Compressing correlated randomness is a key component of secure computation protocols in the silent preprocessing model and has received significant attention in recent years. Most existing constructions target additive correlations, where the parties receive additive shares of a relation. Beyond additive correlations, much less is known: more general linear correlations can be compressed using pseudorandom secret sharing, generic constructions for richer linear-output correlations can be obtained by combining LWE-based multiparty HSS with share conversion, in regimes where the share-conversion overhead is efficient, and general correlations can be compressed using indistinguishability obfuscation.

In this work, we advance the study of compression for useful forms of correlated randomness beyond additive output sharing (concretely, *linear-output* correlations, together with a constrained variant capturing random permutations), without resorting to $i\mathcal{O}$, introducing new techniques, new feasibility results, and relying on more diverse assumptions. We provide several constructions for pseudorandom correlation generators and functions using direct HSS- and LPN/MQ-based techniques. Concretely, we focus on two types of correlations in this work.

First, we introduce pseudorandom *permutation* functions, where each party obtains a portion of a pseudorandom permutation π over the set $\{1, \ldots, n\}$, such that a single party learns nothing about the full permutation beyond its own output. We give a three-party construction of a pseudorandom correlation function (PCF) for permutations from homomorphic secret sharing satisfying a special share programmability property. This construction can be instantiated from a variety of standard assumptions and can even be concretely efficient (generating pseudorandom permutations in a fraction of a second). We describe two applications of pseudorandom permutation functions: one in anonymous broadcast and the other in single secret leader election.

Second, we introduce pseudorandom correlation generators for classical correlations (such as Beaver triples) where additive secret sharing is replaced with *threshold* secret sharing. We obtain pseudorandom correlation functions for degree-1 Shamir shares of NC^1 correlations from a variety of standard assumptions. We also introduce two new PCG constructions for threshold-shared constant-degree correlations in the regimes where either the privacy threshold t or $n - t$ is a constant. These constructions rely on the conjunction of the LPN and MQ assumptions.

Keywords. pseudorandom correlation functions, pseudorandom correlation generators, multi-party permutations, threshold secret sharing.

[1] {couteau,nikolas}@irif.fr
[2] alexander.koch@secorvo.de, work done while the author was at IRIF.
[3] peter.scholl@cs.au.dk
[4] 3s@mit.edu, work done while the author was at MIT.
[5] yexx23@mails.tsinghua.edu.cn

1 Introduction

Modern secure multi-party computation protocols are generally split into two phases: an input-independent offline phase and an input-dependent online phase. In the offline phase, parties generate preprocessing material, that takes the form of correlated randomness. This correlated randomness is stored by the parties for use in the online phase. In the online phase, the parties consume the generated correlated randomness to aid in the computation of the function. This model significantly improves the efficiency of the online phase (reducing communication and rounds of interaction between parties) at the cost of computing a preprocessing phase.

Starting with the work of Boyle et al. [17,19,21], a new paradigm called *silent secure computation* was put forth to improve significantly the efficiency of the offline phase (namely the generation of correlated randomness) with the help of *pseudorandom correlation generators* (PCGs) and *pseudorandom correlation functions* (PCFs) [20],[6] two primitives which allow parties to locally expand a short seed into a large amount of correlated *pseudo*randomness. This paradigm enables efficient deployments of multi-party computation protocols in the preprocessing model by significantly reducing the availability and interaction overheads incurred on the parties in the preprocessing step.

In recent years, many constructions of PCGs and PCFs have been introduced, for various types of useful pseudorandom correlations such as oblivious transfer (OT) correlations [18,19] and Beaver triple correlations [14,21]. These constructions have in turn enabled improved designs of useful secure computation protocols such as private set intersection [30,66,68], threshold signatures [3,49,61], or privacy-preserving machine learning [59,67]. These constructions even extend beyond secure computation, impacting related primitives such as zero-knowledge proofs [10,46,74–76].

1.1 Beyond additive output sharing

The type of correlated randomness that best benefits a secure computation protocol is application-dependent, and an important research effort has been devoted to diversifying the kind of correlations that PCGs and PCFs can produce. To date, we have constructions for OT correlations, Beaver triples, vector oblivious linear evaluations [17], inner-product correlations, matrix multiplication triples [21], one-time truth tables [19], general NC^1 additive correlations [65], unit-vector correlations [6], and more. Most of these constructions produce *additively shared* output correlations. In a 2-party additive correlation, the players get random vectors $\mathbf{x}$ and $\mathbf{y}$ (over some ring) respectively, together with *additive shares* of $f(\mathbf{x}, \mathbf{y})$, where f is some function computable by a polysize circuit over the ring.

Additive sharing is not, however, an absolute feasibility barrier. Gilboa and Ishai [53] showed how PRFs can be used to compress more general linear correlation patterns, an approach later formulated explicitly as generalized pseudorandom secret sharing [11]. Moreover, standard share-conversion techniques allow replicated secret shares to be locally converted into shares under any linear secret-sharing scheme for the same access structure, including Shamir sharing [43]. Combined with generic multiparty additive HSS from LWE [47], this gives an LWE-based feasibility route to richer linear output sharings whenever the share-conversion overhead is efficient, in particular in the threshold regimes considered here. Indistinguishability obfuscation gives still more general constructions for reusable correlated randomness and PCFs [4,47,57]. However, the landscape beyond additive correlations remains restricted, and relies on heavy tools such as LWE-based multiparty HSS or iO. Our goal in this work is to enrich the study of compressing correlations beyond the additive setting, targeting in particular the settings where LWE-based solutions demonstrate the feasibility of iO-less constructions, by introducing new techniques and new direct constructions for specific useful output structures, with different assumption, efficiency, and functionality tradeoffs from these generic routes.

It will be convenient to refer informally to constructions whose outputs are encoded under a fixed linear secret-sharing scheme, rather than additive sharing, as *linear-output* constructions; see Definition 10 for a formal definition of the corresponding class of correlations.

[6] Roughly speaking, a PCF can be used to generate a fresh sample from the target correlation "on the fly" via an evaluation algorithm whereas PCGs require expanding all correlations in a single monolithic evaluation.

A recent result of Agarwal et al. [6] provides one example in this direction: their distributed point function with degree-1 Shamir-shared outputs, when plugged into the LPN-based DPF-to-PCG compiler of Boyle et al. [19], yields PCGs for degree-1 Shamir shares of constant-degree correlations. What remains much less understood is how to obtain direct PCG and PCF constructions for richer linear-output correlations under more specialized assumptions, and how to realize additional constraints within such representations. For example, our three-party permutation construction can itself be viewed as a linear-output construction over $\mathbb{F}_3$, but it additionally requires the shared coefficient to lie in $\{0, 1\}$ so that, after a fixed local transformation, the three outputs are guaranteed to be distinct and hence form a permutation (see Remark 12 for the precise formulation: after the substitution $\nu := 2 - \mu$, the permutation correlation is exactly a linear-output correlation over $\mathbb{F}_3$ whose target distribution is supported on $\mathbb{F}_3^*$).

This leaves many natural target correlations for which improved constructions would be useful in secure computation protocols. Perhaps the most natural example is the generalization of additive correlations to *threshold shared* correlations, where n parties receive random vectors $(\mathbf{x}_1, \cdots, \mathbf{x}_n)$ together with shares of $f(\mathbf{x}_1, \cdots, \mathbf{x}_n)$, but where the output is shared using a linear threshold secret sharing instead of an additive sharing. In the terminology above, these are linear-output correlations (Definition 10) where the output sharing Σ is a threshold LSSS. PCGs and PCFs for such variants would immediately imply silent secure computation protocols tolerating a broader class of corruption thresholds. This would allow, for example, to lower the corruption threshold in secure computation protocols by tolerating a number of failures by the participants, and make protocols more resilient to errors.

Other natural motivations for studying correlations beyond additive output sharing include generating garbled circuit correlations (one party gets a random garbled circuit, and the other party gets the corresponding key) and generating random permutations. Constructions for these types of correlations would have applications to low-communication secure computation, secret leader elections, and committee selection protocols.

1.2 Our contributions

In this work, we study pseudorandom correlation generators and functions beyond additive output sharing. Our focus is on obtaining direct PCG and PCF constructions for specific but natural target correlations, without relying on the generic LWE- or indistinguishability-obfuscation-based routes described above. More specifically, we focus on two correlations:

- *random permutations*, where for a random permutation π over $\{1, \cdots, n\}$, the correlation securely distributes, to each party i, the value $\pi(i)$; and
- *threshold-shared correlations*, where the correlation securely distributes linear threshold shares of a secure computation output over pseudorandom inputs (e.g., threshold shares of a Beaver triple).

In both cases, through different techniques, we obtain non-trivial constructions from assumptions used in the HSS and PCG literature. We summarize our results and compare them to the state of the art in Table 1.

Concretely, we show the following:

3-party pseudorandom permutations. We first introduce pseudorandom permutation generators (PPGs) and functions (PPFs) in the 3-party setting. From now on, by a "pseudorandom permutation (correlation)" for three parties we mean the joint distribution obtained by first sampling a permutation $\pi \xleftarrow{R} S_3$ uniformly and then giving party $i \in \{0, 1, 2\}$ the value $r_i = \pi(i)$. A PCF implements this correlation if the joint output (r_0, r_1, r_2) is computationally indistinguishable from $(\pi(0), \pi(1), \pi(2))$ for a uniform π. Each marginal r_i is uniform on $\{0, 1, 2\}$. Any two parties can recover π, while one party learns nothing beyond its own r_i.[7]

Our main construction relies on two ingredients: homomorphic secret sharing for restricted-multiplication straightline (RMS) programs with simulatable memory shares, and a PRF whose

[7] This is *not* the block-cipher notion of a keyed PRP. Here the permutation is sampled at correlation time and only revealed per-coordinate.

evaluation is in NC^1. The required HSS schemes are known from a variety of assumptions, including DCR [65,69], the hardness of the Joye–Libert encryption scheme, the DDH and DXDH assumptions over class groups, the Hard Subgroup Membership assumption over class groups [2], and LWE with super-polynomial modulus-to-noise ratio [29]; see also Couteau et al. [42] for the simulatable-memory property of these schemes. Both ingredients can be instantiated jointly from LWE with super-polynomial modulus-to-noise ratio and from DCR. For LWE, the required HSS follows from Boyle, Kohl, and Scholl [29], while Banerjee, Peikert, and Rosen [9] construct PRFs computable in NC^1 from LWE in the corresponding large-modulus, low-noise regime. For DCR, the HSS instantiations use RSA moduli that are Blum integers, while the factoring-based PRF of Naor and Reingold [63] applies to arbitrary efficiently samplable distributions of Blum integers and is computable in $\mathsf{TC}^0 \subseteq \mathsf{NC}^1$. Since factoring such moduli breaks DCR, DCR therefore also provides both ingredients. For the Joye–Libert- and class-group-based HSS instantiations, we additionally require a secure PRF computable in NC^1.

While our main focus is on feasibility, we additionally show that by replacing the PRF with a suitable low-complexity (weak) PRF, our construction can actually achieve good concrete efficiency.

We provide a candidate concrete instantiation for the weak PRF, and estimate that generating a pseudorandom permutation takes less than 100ms on one core of a standard laptop. A single corrupted party learns nothing beyond its own output; this is the strongest non-trivial privacy guarantee possible for a 3-party permutation, since any two parties necessarily determine the full permutation.

As a secondary contribution, we also describe a 3-party PPG from the hardness of the quasi-abelian syndrome decoding problem, an assumption recently introduced by Bombar et al. [14]. Compared with our main construction, the PPG construction has a few downsides: (1) it is only a PPG (hence all permutations are generated in a one-shot fashion, which limits its usefulness in applications), and (2) it generates a *biased* pseudorandom permutation, where each party can guess the indices received by the other two with probability 2/3. From a feasibility perspective, biased three-party permutation correlations can also be obtained from the general LPN-based PCG framework of Boyle et al. [19]; our QA-SD construction instead gives a more concrete instantiation of the programmable template used here. Despite the bias, the construction can still be used directly for single secret leader election, and in the multi-round anonymous-broadcast setting the bias can be removed using a simple randomness extractor, as discussed in Section 6.

Degree-1 Shamir-shared correlations. As a direct byproduct of the techniques introduced for our pseudorandom permutation function, we obtain pseudorandom correlation functions for distributing degree-1 Shamir shares [70] of any relation computable by an RMS program (a class of programs that captures in particular all NC^1 functions), from the same assumptions.

Compared to the recent work by Agarwal et al. [6], which shows how to construct distributed point functions outputting degree-1 Shamir shares from OWFs and yields (using the LPN-based DPF-to-PCG compiler of Boyle et al. [19]) a PCG for degree-1 Shamir shares of constant degree correlations, our result is incomparable: we rely on a stronger assumption but obtain a pseudorandom correlation *function* rather than generator, for a much larger class of functions.

Shamir-shared correlations when $n-t$ is constant. Building upon the recent work of Couteau et al. [41], we construct a pseudorandom correlation generator for degree-t Shamir shares of constant-degree correlations when $n-t$ is constant. Our construction is based on the conjunction of the learning parity with noise (LPN) and multivariate quadratic (MQ) assumptions.

Threshold-shared correlations for constant t. We also provide a different approach to build a PCG for constant-degree correlations with a linear threshold sharing secure against t corruptions, for any constant t, from the same assumptions (LPN and MQ). In contrast with the previous construction, however, the underlying linear threshold secret sharing scheme is not Shamir secret sharing. Nevertheless, it enjoys essentially the same applications (as any linear secret sharing with the same threshold can usually be used as a drop-in replacement in MPC applications using Shamir secret sharing).

Our results do not resolve the problem of constructing direct PCGs and PCFs for richer output structures in full generality. The permutation construction is restricted to three parties, and our threshold-sharing constructions apply in regimes where either t or $n-t$ is constant. Nevertheless,

Primitive	Assumption	Target Correlation	# parties
PCF/PRSS [11,53]	PRF	Linear correlations (uniform-target linear-output, Remark 12)	n
HSS + share conversion [43,47]	LWE	Linear-output correlations (any poly-time samplable target)	n
PCF [47,57]	$i\mathcal{O}$	Any reverse-samplable	2
PCF [4]	multi-key FHE + $i\mathcal{O}$	Any reverse-samplable	n
PCF [4]	multi-key FHE + $i\mathcal{O}$ + ROM	Any	n
PCG [6]+ [19]	LPN	Degree-1 Shamir shares of const. degree poly	n
PCG [19]	LPN	Biased 3-party permutations (generic feasibility)	3
	This work		
PCF (Section 4)	HSS+PRF in NC^1	Permutations	3
PCF (Section 4)	HSS+PRF in NC^1	Degree-1 Shamir shares of NC^1 functions	n
PCG (Section 5)	LPN+MQ	Degree-t Shamir shares of const. degree poly ($n - t = O(1)$)	n
PCG (Section 5)	LPN+MQ	Linear threshold shares of const. degree poly ($t = O(1)$)	n
PCG (Appendix B)	Quasi-Abelian SD	Permutations (biased)	3

Table 1: Selected constructions relevant to the correlation classes considered in this work. Colors highlight results that do not require $i\mathcal{O}$. "Linear-output" refers to correlations whose outputs are encoded under a fixed linear secret-sharing scheme rather than additive sharing; see Definition 10 for the formal definition. Reverse-samplable means that, given the corrupt parties' outputs of a correlation sample, the honest parties' outputs can be efficiently simulated in a way that is indistinguishable from the original sample.

they provide direct constructions with different assumption and functionality tradeoffs from the generic LWE- or $i\mathcal{O}$-based routes described above. To handle these settings, we introduce new ideas, in particular around enforcing share distinctness and programming shares across related correlations. We hope that these techniques will be useful as starting points for future constructions of PCGs and PCFs for richer classes of correlations. In addition, our results already have direct applications, as we outline in Section 1.3.

Future directions. Our constructions show that useful correlations beyond additive output sharing can be obtained directly from a broader range of PCG and HSS techniques. However, our new constructions do come with some limitations: a restriction to 3 parties for our PPF, and a restriction to t or $n - t$ being constant for our PCGs for threshold secret sharing of low-degree correlations. As such, we view our contributions as providing an initial set of results and techniques for a broader study of direct constructions for richer output correlations. Natural directions for future work include extending pseudorandom permutation functions beyond three parties and obtaining threshold-shared PCGs for more general corruption thresholds.

1.3 Applications

While a main focus of our work is on the conceptual contribution—giving direct constructions for richer output correlations under HSS- and LPN-based assumptions—our constructions also have concrete applications. We outline two such applications next.

Pseudorandom permutation functions. We show that PPFs have a natural application in anonymous communication protocols and single secret leader election. We find that we can replace the expensive "offline" preprocessing protocol of Studholme and Blake [71] (which is used to generate pseudorandom permutations) with a PCF, to instantiate a Dining-Cryptographers network for anonymous broadcast with one uniquely assigned slot per party. In particular, we show that using a PCF, each party can locally determine where to write their message in a bulletin board, in each round. Here, the pseudorandomness of the permutation guarantees that no other party learns the association between messages and the other parties and the permutation itself guarantees each party writes to a uniquely assigned "slot" removing the need for redundancy (a common solution used in prior work [38,56]). We also show how a PCF for pseudorandom permutations enables a non-interactive single secret leader election (SSLE) protocol, where the leader is elected by having the lowest permutation value. In turn, SSLE has applications to proof-of-stake cryptocurrencies and other problems in distributed systems [15].

PCGs for threshold secret sharing. Plugging our new PCGs in existing secure computation protocols in the correlated randomness model, we obtain n-party secure computation protocols with *silent preprocessing* (concretely, the preprocessing phase can be implemented with $\mathsf{poly}(n)\cdot\mathsf{polylog}(s)$ communication for generating length-s correlations) secure against t corruptions in either of the two regimes $t = O(1)$ or $n-t = O(1)$, under the LPN and MQ assumptions. Within the LPN-based PCG framework, prior constructions naturally produced additive output sharings; our constructions extend this to these two threshold regimes. The regime $n - t = O(1)$ is especially appealing, as it translates to MPC protocols with silent preprocessing secure against adversaries corrupting almost all of the parties, while remaining resilient to a small (constant) number of failures: reconstruction from degree-t Shamir shares requires $t + 1$ participants and therefore tolerates up to $n - t - 1$ failures. Such resiliency to failures is crucial in applications where a failure to produce an output can have important consequences, e.g., in threshold signatures (and indeed, the use of PCGs for additive correlations has been pointed out as a significant limitation of [3]).

The work of [19] also showed that n-party PCGs for constant-degree correlations imply secure computation with almost-sublinear communication $n \cdot s/c$ for circuits of size s and any constant c. Plugging our PCGs in their compiler directly implies the same result in the regime $n - t = O(1)$, with security against t corruptions and resiliency to up to $n - t - 1$ failures.

Both applications (to silent MPC and to sublinear MPC) follow immediately by plugging our new PCGs into existing compilers from previous works. Therefore, we refrain from providing a detailed treatment of these applications in the main body of this work and will only note their existence here.

2 Technical Overview

In this section, we provide a detailed technical overview of our approach and constructions. In Section 2.1, we briefly recall the syntax of PCGs and PCFs, as well as the HSS tools used throughout the paper. In Section 2.2, we first describe our main building block, a generic way to obtain HSS with degree-1 Shamir-shared outputs from two-party HSS schemes satisfying a simulatable-memory property. We then explain how this building block yields PCFs for degree-1 Shamir-shared correlations, and how, in the three-party setting, it can be specialized to obtain PCFs for permutation correlations. Finally, in Section 2.3, we describe our two approaches for higher-threshold PCGs: one for the regime where $n - t$ is constant, based on replicated secret sharing, and one for the regime where t is constant, based on party emulation.

As discussed in the introduction, generic feasibility results for richer *linear-output* correlations (Definition 10) are already available from PRF- and LWE-based techniques. In the terminology of Section 3.5, pseudorandom secret sharing covers linear-output correlations whose target distribution is *uniform*; some constructions of this work target *structured* target distributions, such as distributions with restricted support, as for permutations (Remark 12). The constructions below target more specific assumption and functionality tradeoffs. On the PCF side, the main question is how to turn two-party HSS with additive reconstruction into an n-party HSS whose outputs form a single degree-1 Shamir sharing. On the PCG side, the difficulty is to obtain higher-threshold output sharings while using the available LPN- and MQ-based multiparty DPFs, whose useful parameter regime here has only a constant number of parties. The two threshold constructions address this latter limitation in complementary parameter regimes.

2.1 Background

We start by giving a brief overview of PCGs and PCFs to aid in understanding of the technical overview. See Section 3 for formal definitions. A PCG for a target two-party correlation $\mathcal{C}$ consists of two algorithms:

- $\mathsf{Gen}(1^\lambda) \to (\mathsf{k}_0, \mathsf{k}_1)$: a randomized algorithm that generates a pair of short, correlated seeds $(\mathsf{k}_0, \mathsf{k}_1)$ given a security parameter λ, and
- $\mathsf{Expand}(\mathsf{k}_\sigma) \to R_\sigma$: a deterministic algorithm which stretches a seed k_σ into a long output string R_σ.

Remark 1 (PCFs). In the case of a PCF, Expand is replaced with an evaluation algorithm Eval that takes an additional input x and outputs a single instance of the correlation (computed as a function of x).

The security requirement for a PCG is that the joint outputs (R_0, R_1) should be indistinguishable from $\mathcal{C}$, not only to external parties but also to each of the parties that possesses only one of the seeds. PCFs have an analogous security requirement with the main difference being on-demand evaluation of R_0 and R_1.

Our constructions will rely on homomorphic secret sharing (HSS). Informally, an n-party HSS for a class of functions $\mathcal{F}$ is a pair of algorithms $(\mathsf{Share}, \mathsf{Eval})$ where $\mathsf{Share}(x)$ distributes n shares $(x_1, \cdots, x_n)$ of an input x, and for any $f \in \mathcal{F}$, $\mathsf{Eval}(i, x_i, f)$ produces a share y_i of $f(x)$—that is, $\mathsf{Rec}(y_1, \cdots, y_n) = f(x)$, where Rec is the reconstruction procedure for the output shares. Homomorphic secret sharing with additive reconstruction are known for the class of RMS programs (a superclass of NC^1) from a variety of standard assumptions [2, 22, 25, 29, 65, 69] in the 2-party setting, and for low-depth circuits in the 4-party [23, 35] and 8-party [40] setting. In this work, we will be interested in the setting where the outputs are shared via richer linear secret-sharing schemes, and in particular via degree-t Shamir sharing [70] — i.e., in *linear-output* HSS (Definition 10).

2.2 The degree-1 Shamir-output HSS building block and its applications

The main technical building block behind our PCF constructions is a way to obtain HSS with degree-1 Shamir-shared outputs from existing two-party HSS schemes. After homomorphic evaluation of a function f, the parties obtain evaluations of a degree-1 polynomial whose secret is $f(x)$. Any single party learns nothing about the output, while any two parties can reconstruct it. This is the linear-output HSS (in the sense of Definition 10) at the core of our PCF results.

From two-party HSS to degree-1 Shamir-output HSS. The most expressive known HSS constructions support evaluation of large classes of functions, including arbitrary polynomial-size circuits from LWE [35, 47], and restricted-multiplication straightline programs, a class containing NC^1, from group-theoretic assumptions such as DDH, DCR, and assumptions over class groups [2, 25, 65, 69]. These schemes, however, naturally give additively shared outputs.

We show that many two-party HSS schemes for restricted-multiplication programs can be generically lifted to an n-party scheme with degree-1 Shamir-shared outputs. The key property we use is *simulatable memory shares*, introduced by Couteau et al. [42].

Informally, in the relevant HSS schemes, the intermediate values of the computation are represented by additive "memory shares". The simulatable-memory property says that one party's memory share can be sampled independently of the underlying value, while the other party's memory share can later be programmed so that the pair encodes the desired value.

We exploit this property as follows. To share an input x, we first generate ordinary two-party HSS shares of x. Party P_0 receives the first HSS share, while every other party P_i, for $i \in \{1, \ldots, n-1\}$, receives the second HSS share. Thus, for each pair (P_0, P_i), the two parties can run the underlying two-party HSS evaluation and obtain additive shares of $f(x)$.

The goal, however, is not to give each pair an unrelated additive sharing. Instead, we want the outputs of all parties to lie on a single degree-1 polynomial whose value at zero is $f(x)$, as in standard Shamir sharing. We associate party P_i with the Shamir evaluation point $\xi_i = i + 1$. Let y_A denote the output of P_0 in the underlying two-party extended evaluation. For each $i \geq 1$, we use the simulatable-memory property to program the scalar

$$\alpha_i = \frac{i}{i+1}.$$

If $y_B^{(i)}$ is the corresponding output of P_i, extended-evaluation correctness gives

$$y_A - y_B^{(i)} = \frac{i}{i+1} f(x).$$

Party P_0 keeps y_A as its output, while party P_i locally multiplies its output by $i + 1$.

The resulting values are evaluations at $1, \ldots, n$ of the polynomial

$$q(z) = f(x) + z\big(y_A - f(x)\big).$$

Indeed, $q(1) = y_A$, and for $i \geq 1$,

$$q(i+1) = (i+1)y_A - if(x) = (i+1)y_B^{(i)}.$$

In particular, $q(0) = f(x)$, so these are degree-1 Shamir-form shares of the evaluated function under the standard convention. When the underlying HSS output shares are pseudorandom, as in the output-randomized form used in our constructions, the resulting sharing is computationally indistinguishable from a standard random degree-1 Shamir sharing. Instantiating the underlying two-party HSS for NC^1 or RMS programs therefore gives an n-party HSS with degree-1 Shamir-shared outputs for the same class of functions, under the corresponding assumptions.

PCFs for degree-1 Shamir-shared correlations. This HSS building block immediately yields PCFs for degree-1 Shamir-shared correlations. At key-generation time, the parties receive HSS shares of a seed for a pseudorandom sampler. On input x, they homomorphically evaluate a PRF, and then evaluate the sampling circuit for the desired correlation using the resulting pseudorandomness. If the sampler is computable by an RMS program, and in particular if it is computable in NC^1, the output is obtained directly as a degree-1 Shamir sharing.

This gives a PCF, rather than only a PCG as the parties can evaluate the correlation on demand at any input x, without expanding an entire batch of correlations at once. This is the first application of the degree-1 Shamir-output HSS transformation.

Specializing the building block to three-party permutations. The same building block also gives a PCF for three-party permutation correlations, but here the connection is slightly less direct and uses a small algebraic observation. A permutation of $\{0,1,2\}$ can be generated from a degree-1 Shamir sharing of a bit over $\mathbb{F}_3$, where the bit is encoded as the coefficient of the sharing polynomial rather than as its constant term.[8]

Concretely, let $x \in \mathbb{F}_3$ be uniform and let $\mu \in \{0,1\}$. Consider the polynomial

$$p(y) = \mu y + x$$

and its evaluations

$$p(0) = x, \qquad p(-1) = x - \mu, \qquad p(1) = x + \mu.$$

From these three values, the parties can locally derive the tuple

$$x, \qquad x + 2 - \mu, \qquad x + 1 + \mu.$$

These three elements are always pairwise distinct in $\mathbb{F}_3$: if the first two were equal, then $\mu = 2$; if the first and third were equal, then $\mu + 1 = 0$; and if the last two were equal, then $2\mu = 1 \pmod 3$. All three are impossible for $\mu \in \{0,1\}$. Thus the tuple is a permutation of $\mathbb{F}_3$. In the language of Definition 10, the permutation correlation is thus exactly a linear-output correlation over $\mathbb{F}_3$ whose target distribution is supported on $\mathbb{F}_3^*$; see Remark 12.

Therefore, to obtain a PCF for three-party permutations, it suffices to generate a pseudorandom bit μ inside the three-party degree-1 Shamir-output HSS scheme and then apply the above local conversion. We do this by sharing a PRF key using this HSS construction and homomorphically evaluating a one-bit PRF over $\mathbb{F}_3$. The resulting Shamir-shared bit is then mapped to a permutation as above.

For a single corrupted party, security follows from the security of the underlying HSS, the pseudorandomness of the PRF, and the privacy of the degree-1 Shamir sharing. This is the strongest non-trivial privacy guarantee for a three-party permutation, since any two parties necessarily determine the full permutation.

A weaker PCG construction from a variant of LPN. We also give an alternative construction of a PCG for three-party permutations, relying on the quasi-abelian syndrome decoding problem

[8] This is sometimes called the value at the point at infinity. Using this representation is convenient over $\mathbb{F}_3$, since it lets us use the three evaluation points $0, -1, 1$.

over $\mathbb{F}_3$, a variant of LPN introduced in [14]. This construction uses the same permutation encoding described above, but replaces the HSS-based generation of an unbiased Shamir-shared bit by a PCG for biased secret-shared bits modulo 3.

The resulting random bits have bias 1/6, and therefore yield only a biased three-party permutation PCG. Nevertheless, this construction is significantly more concretely efficient than the HSS-based PCF, and the techniques may be useful in future improvements. We defer the details of this construction to Appendix B. From a feasibility viewpoint, biased three-party permutation correlations can also be obtained from the general LPN-based PCG framework of Boyle et al. [19]; the purpose of our QA-SD construction is instead to obtain a more concrete instantiation of the programmable template described above.

2.3 PCGs for threshold-shared correlations

We now describe our PCGs for threshold-shared correlations with higher privacy thresholds. Both constructions build on the DPF-to-PCG paradigm of Boyle et al. [19], which shows how to obtain PCGs for low-degree correlations from distributed point functions and LPN.

Starting point. A distributed point function is a primitive that distributes shares of a point function $f_{\alpha,\beta}$ to n parties (a point function $f_{\alpha,\beta}$ satisfies $f(\alpha) = \beta$ and evaluates to 0 on all $x \neq \alpha$) such that given any input x, the parties can locally compute shares of $f_{\alpha,\beta}(x)$. The pseudorandom correlation generator originally introduced by Boyle et al. [19] can be viewed as a general compiler that, given an n-party DPF over a domain of size $O(N^d)$ and assuming LPN, provides an n-party PCG for degree-d N-variate correlations whose seed length is dominated by $O(\lambda^d)$ DPF keys. State-of-the-art PCGs are instantiations of this template with the PRG-based 2-party additive DPF of Boyle et al. [26]. In this section, we describe extensions of this template to threshold-shared constant-degree correlations.

A first observation. The DPF-to-PCG compiler suggests a natural route to threshold-shared PCGs. First build DPFs whose outputs are themselves threshold shares, and then plug them into the LPN-based compiler of [19]. A simple way to obtain such a DPF is to start from an additive DPF for $\binom{n}{t}$ parties. Associate one additive DPF key with each subset $T \subseteq [n]$ of size t, and give this key to all parties outside T. This gives a replicated sharing of the DPF output, which can then be locally converted to degree-t Shamir shares. When t is constant, the number $\binom{n}{t}$ is polynomial in n, so this transformation is efficient assuming the required additive multiparty DPF is available.

The obstacle, if we want to stay within the LPN- and MQ-based DPF framework, is that the available constructions do not give additive DPFs for an arbitrary polynomial number of parties with the parameters needed here. Generic LWE-based techniques provide a broader feasibility route, as discussed in the introduction, but do not give the assumption tradeoff targeted by these constructions. The recent work of Couteau et al. [41] gives multiparty DPFs from LPN and MQ, but only for a constant number of parties, or with a blowup that becomes nontrivial only for very small parameter ranges. Our two constructions deal with this limitation in complementary ways. When $n - t$ is constant, we use replicated secret sharing so that the DPFs required for the cross-terms involve only a constant number of parties. When t is constant, we instead use party emulation to reduce the desired threshold structure over n real parties to a degree-1 Shamir-output structure over $m = \binom{n}{t}$ virtual parties.

First construction: constant $n - t$, polynomial n. For this construction, we assume that we have at our disposal a DPF with degree-t Shamir-shared outputs for constant n and t. As outlined above, this can be constructed generically given any $\binom{n}{t}$-party additive DPF, which in turn can be based on LPN and MQ [41]. For simplicity, assume that we want to generate Shamir shares of a Beaver triple: the n parties should receive (many) degree-t Shamir shares of pseudorandom triples $(a, b, a \cdot b)$. Our PCG proceeds as follows: first, for every subset S of size t, generate pseudorandom values a_S, b_S and distribute them to all parties not in S. This can be done succinctly by generating many a_S's and b_S's (for a fixed S) as LPN samples, by dealing the corresponding noise vector to all parties outside of S. Concretely, we use the standard LPN-based template of [19] to generate each pseudorandom share: let $\mathbf{e}_S, \mathbf{f}_S$ denote random noise vectors (i.e., sparse vectors) and fix a public random compressing matrix H. Define $\mathbf{a}_S \coloneqq H \cdot \mathbf{e}_S$ and $\mathbf{b}_S \coloneqq H \cdot \mathbf{f}_S$. Under the LPN assumption,

$(\mathbf{a}_S, \mathbf{b}_S)$ are indistinguishable from random. Furthermore, every party outside S can reconstruct $\mathbf{a}_S$ and $\mathbf{b}_S$ from the sparse vectors $\mathbf{e}_S, \mathbf{f}_S$, which can be represented succinctly. Now, we leverage a classical observation from [21]: all component-wise products $\mathbf{a}_S \odot \mathbf{b}_T$ for subsets S, T can be read from the diagonal of $\mathbf{a}_S \cdot \mathbf{b}_T^\intercal = H \cdot (\mathbf{e}_S \cdot \mathbf{f}_T^\intercal) \cdot H^\intercal$, and each entry can be reconstructed given shares of $(\mathbf{e}_S \cdot \mathbf{f}_T^\intercal)$ (assuming the secret sharing scheme is linear). In turn, because $\mathbf{e}_S, \mathbf{f}_T$ are sparse (weight-w), so is their tensor product (weight-w^2), hence they can be expressed as a sum of w^2 point functions (evaluated on their entire domain).

Now, define $a = \sum_S a_S$ and $b = \sum_S b_S$. We have $a \cdot b = \sum_{S,T} a_S \cdot b_T$. To share $a \cdot b$, we need to share all terms $a_S \cdot b_T$ for all pairs of t-sized subsets S, T. To share $a_S \cdot b_T$ for fixed S, T, it suffices to distribute the corresponding cross-term among the parties in

$$M_{S,T} \coloneqq ([n] \setminus S) \cup ([n] \setminus T),$$

namely the parties that know at least one of the two sparse vectors. Thus we use w^2 DPFs with degree-$(m - (n-t))$ Shamir-shared outputs, where $m = |M_{S,T}|$, one for each nonzero entry of $\mathbf{e}_S \otimes \mathbf{f}_T$. Fortunately, m and $n - t$ are constants, hence this can be achieved using the DPF from LPN+MQ outlined above. The full construction involves an $\binom{n}{n-t}^2$ overhead for generating the shares of all cross-terms. Generalizing the construction to arbitrary degree-d correlations (for constant d) is straightforward, at the cost of increasing the overhead to $\binom{n}{n-t}^d$.

In our next construction, we handle the complementary case where t is a constant. Note that our first construction does not generalize to this setting even though there is a polynomial number of subsets S of t parties, because it requires a DPF with degree-$(m-(n-t))$ Shamir-shared outputs, and m and $(n-t)$ would not be constant anymore. As such, a different strategy is required.

Second construction: polynomial n and constant t. The second construction handles the complementary regime where t is constant and n may be polynomial. Here the replicated-sharing approach above no longer suffices because although there are only polynomially many subsets of size t, the cross-term distribution would require DPFs for a non-constant number of parties. We therefore use a party-emulation approach, inspired by the threshold-emulation framework of Cohen et al. [37] and Hirt and Maurer [58].

The idea is to create one virtual party for every subset $T \subseteq [n]$ of size t, so that the number of virtual parties is $m = \binom{n}{t}$.

We then use a DPF with degree-1 Shamir-shared outputs over these virtual parties. The reason this helps is because if an adversary corrupts at most t real parties, then it can fully reconstruct the state of at most one virtual party, namely the virtual party indexed by the corrupted set itself. So privacy against t corruptions among the real parties reduces to privacy against one corrupted virtual party.

It remains to explain how the real parties emulate the virtual ones. For each virtual party T, its DPF key is shared among the real parties in T using a t-party additive HSS scheme. Each real party participates in the emulation of all virtual parties whose index set contains it, and homomorphically evaluates the DPF evaluation procedure on its shares of the corresponding virtual keys. The outputs of these emulated evaluations form a composed linear threshold sharing over the real parties.

To instantiate this, we combine two ingredients. First, we use the degree-1 Shamir-output DPF of Agarwal et al. [6]. Using the arity-reduction technique of Couteau and Kumar [40], this DPF can be made computable by constant-degree polynomials under MQ. Second, we use the LPN-based DPF-to-HSS compiler of Couteau et al. [41] to obtain the additive HSS needed for the outer emulation. The resulting DPF can then be plugged into the Boyle et al. DPF-to-PCG compiler [19] to obtain PCGs for constant-degree correlations.

The price we pay for this approach is that the final sharing is not Shamir's secret sharing. Rather, it is a composed linear threshold sharing private against t corruptions because the virtual outputs are first shared using degree-1 Shamir sharing over the virtual parties, and each virtual share is then additively shared among the real parties that emulate it. This is sufficient for the PCG construction, but it should not be described as a DPF with degree-t Shamir-shared outputs. We refer the reader to Section 5.5 for the full details.

3 Preliminaries

Notation. We let $\mathbb{N}$ denote the set of natural numbers. We denote a vector $\mathbf{u}$ using bold lowercase letters and let $\mathbf{u}[i]$ denote the i-th coordinate of $\mathbf{u}$. We denote by $\mathsf{poly}(\cdot)$ the set of all polynomials and by $\mathsf{negl}(\cdot)$ any negligible function. We let $x \xleftarrow{R} S$ denote a uniformly random sample drawn from S. We let $x \leftarrow \mathcal{A}$ denote assignment from a (possibly randomized) algorithm $\mathcal{A}$ and $x := y$ denote initialization of x to the value of y. By an *efficient* algorithm $\mathcal{A}$ we mean that $\mathcal{A}$ is modeled by a (possibly non-uniform) Turing Machine that runs in probabilistic polynomial time. We write $D_0 \approx_c D_1$ to mean that two distributions D_0 and D_1 are *computationally* indistinguishable to all efficient distinguishers $\mathcal{D}$ and $D_0 \approx_s D_1$ to mean that D_0 and D_1 are *statistically* indistinguishable.

Threshold convention. When discussing threshold sharing and security, we use t to denote the privacy threshold: any set of at most t shares should reveal no information about the secret, while reconstruction requires at least $t+1$ shares. In particular, a degree-t Shamir sharing is private against t shares and reconstructs from any $t+1$ shares. We use the degree/privacy threshold rather than the "t-out-of-n" terminology in order to avoid ambiguity about whether t denotes the privacy or reconstruction threshold.

3.1 Homomorphic Secret Sharing

Homomorphic Secret Sharing (HSS) [25] can be seen as a distributed analogue to homomorphic encryption. In contrast to the centralized model where a single server computes on encrypted inputs, in HSS, shares of the input are distributed among multiple parties, who then compute a share of the output through local homomorphic computations. Here, we use a threshold variant of HSS, similar to the definition from [28].

Definition 2 (Homomorphic Secret Sharing [25]). *Let λ be a security parameter and $\mathcal{R}$ be a finite ring. Let $t \in \mathbb{N}$ be the privacy threshold and $n \in \mathbb{N}$ be the number of parties. An n-party Homomorphic Secret Sharing (HSS) scheme secure against t corruptions for a class of programs $\mathcal{P}$ with input space $\mathcal{I} \subseteq \mathcal{R}$ and share space $\mathcal{S} \subseteq \mathcal{R}$, consists of four efficient (possibly randomized) algorithms $\mathsf{HSS} = (\mathsf{Setup}, \mathsf{Input}, \mathsf{Eval}, \mathsf{Rec})$ such that:*

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{pk}, \mathsf{ek}^i)$. *Takes as input the security parameter λ. Outputs a public key pk and an evaluation key ek^i for each party $i \in [n]$.*
- $\mathsf{Input}(\mathsf{pk}, x) \to [\![x]\!]^i$. *Takes as input the public key pk and an input $x \in \mathcal{I}$. Outputs HSS input shares $[\![x]\!]^i$ for each party i.*
- $\mathsf{Eval}(\sigma, \mathsf{ek}^\sigma, ([\![x_1]\!]^\sigma, \ldots, [\![x_m]\!]^\sigma), P) \to \langle y \rangle^\sigma$. *Takes as input the party identifier σ, an evaluation key ek^σ, a vector of m HSS input shares, and a m-input program $P \in \mathcal{P}$. Outputs a share of the output $y = P(x_1, \ldots, x_m)$ in share space $\mathcal{S}$.*
- $\mathsf{Rec}_T((\langle y \rangle^i)_{i \in T}) \to y$, *where $T \subset [n]$ has size $\geq t+1$. Given a set of at least $t+1$ shares of y, it reconstructs the output y.*

The above functionality must satisfy the following properties:

Correctness. *For all security parameters $\lambda \in \mathbb{N}$, any $n \in \mathbb{N}$, any $T \subset [n]$ with $|T| \geq t+1$, and any m-input program $P \in \mathcal{P}$ with input space $\mathcal{I} \subseteq \mathcal{R}$, it holds that for any $x_1, \ldots, x_m \in \mathcal{I}$:*

$$\Pr\left[y = P(x_1, \ldots, x_m) : \begin{array}{r} (\mathsf{pk}, (\mathsf{ek}^\sigma)_{\sigma \in [n]}) \leftarrow \mathsf{Setup}(1^\lambda) \\ ([\![x_i]\!]^1, \ldots, [\![x_i]\!]^n) \leftarrow \mathsf{Input}(\mathsf{pk}, x_i)\ \forall i \in [m] \\ \langle y \rangle^\sigma \leftarrow \mathsf{Eval}(\sigma, \mathsf{ek}^\sigma, ([\![x_i]\!]^\sigma)_{i \in [m]}, P)\ \forall \sigma \in T \\ y := \mathsf{Rec}_T((\langle y \rangle^\sigma)_{\sigma \in T}) \end{array} \right] \geq 1 - \mathsf{negl}(\lambda).$$

Security. *For all efficient adversaries $\mathcal{A}$, and any $T \subset [n]$, with $|T| \leq t$ there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that:*

$$\Pr\left[b' = b : \begin{array}{r} (x_0, x_1, \mathsf{st}) \leftarrow \mathcal{A}(1^\lambda) \\ (\mathsf{pk}, (\mathsf{ek}^\sigma)_{\sigma\in[n]}) \leftarrow \mathsf{Setup}(1^\lambda) \\ b \xleftarrow{R} \{0,1\} \\ (\llbracket x_b \rrbracket^1, \ldots, \llbracket x_b \rrbracket^n) \leftarrow \mathsf{Input}(\mathsf{pk}, x_b) \\ b' \leftarrow \mathcal{A}(\mathsf{st}, \mathsf{pk}, (\mathsf{ek}^\sigma, \llbracket x_b \rrbracket^\sigma)_{\sigma\in T}) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

HSS memory values. In many existing, two-party HSS schemes [25, 42, 45, 50], we can consider so-called "memory shares" of intermediate values during an HSS evaluation held by party σ, which we denote by $\langle\langle x \rangle\rangle^\sigma$. These memory shares can be added together and multiplied with an HSS input share to produce a new memory share. Also, there is an additional efficient algorithm MemGen that takes as input an index $\sigma \in [n]$, and evaluation key ek_σ and an input $x \in \mathcal{I}$, and outputs a memory share $\langle\langle x \rangle\rangle^\sigma$ of x.

Extended Evaluation. Moreover, for the two party case specifically, HSS schemes support a so-called "extended evaluation", as given by [42, Lemma 1]. Informally, there is an additional evaluation algorithm $\mathsf{ExtEval}$ that is similar to Eval but takes one additional input $M_\sigma = \langle\langle z \rangle\rangle^\sigma$—a memory share of $z \in \mathcal{I}$—and outputs a $\langle y \rangle^\sigma$, such that upon reconstruction $\langle y \rangle^0 - \langle y \rangle^1$ one obtains the z-multiplied output $z \cdot P(x_1, \ldots, x_n)$. We will use this to multiply our outputs by a $z \in \{-1, 1\}$.

More formally, the syntax of $\mathsf{ExtEval}$ is as follows:

- $\mathsf{ExtEval}(\sigma, \mathsf{ek}^\sigma, \langle\langle \alpha \rangle\rangle^\sigma, (\llbracket x_1 \rrbracket^\sigma, \ldots, \llbracket x_m \rrbracket^\sigma), P) \to \langle y \rangle^\sigma$. Takes as input an index σ, an evaluation key ek^σ, a memory share $\langle\langle \alpha \rangle\rangle^\sigma$ of $\alpha \in \mathcal{I}$ and a vector of m HSS input shares, and a m-input program $P \in \mathcal{P}$. Outputs a subtractive share of the output $y = \alpha P(x_1, \ldots, x_m)$.

Correctness of this algorithm means that

$$\Pr\left[y = z \cdot P(x_1, \ldots, x_m) : \begin{array}{r} (\mathsf{pk}, (\mathsf{ek}^A, \mathsf{ek}^B)) \leftarrow \mathsf{Setup}(1^\lambda) \\ (\llbracket x_i \rrbracket^A, \llbracket x_i \rrbracket^B) \leftarrow \mathsf{Input}(\mathsf{pk}, x_i) \forall i \in [m] \\ M^A \leftarrow \mathsf{MemGen}(A, \mathsf{ek}^A, z) \\ M^B \leftarrow \mathsf{MemGen}(B, \mathsf{ek}^B, z) \\ \langle y \rangle^A \leftarrow \mathsf{ExtEval}(A, \mathsf{ek}^A, M^A, (\llbracket x_i \rrbracket^A)_{i\in[m]}, P) \\ \langle y \rangle^B \leftarrow \mathsf{ExtEval}(B, \mathsf{ek}^B, M^B, (\llbracket x_i \rrbracket^B)_{i\in[m]}, P) \\ y \leftarrow \langle y \rangle^A - \langle y \rangle^B \end{array}\right] \geq 1 - \mathsf{negl}(\lambda),$$

i.e., if we run it like Eval, but with the additionally generated memory share of a $z \in \mathcal{I}$, we get a subtractive share of $z \cdot P(x_1, \ldots, x_m)$.

Programmability of HSS memory values. Here, we follow the definition of Couteau et al. [42, Section 4.3].

Definition 3 (HSS with Simulatable Memory Values). *Let* $\mathsf{HSS} = (\mathsf{Setup}, \mathsf{Input}, \mathsf{Eval})$ *be an HSS scheme with input space $\mathcal{I}$. Then,* HSS *is* simulatable w.r.t. its memory values *if there are efficient algorithms* Sim_0, Sim_1, *where* Sim_0 *takes as input 1^λ and outputs a memory value M_0, and* Sim_1 *takes as input a memory value M_0, an element $z \in \mathcal{I}$, and two keys $\mathsf{ek}^A, \mathsf{ek}^B$, and outputs a memory value M_1. Then we want:*

Simulation Correctness. *For any security parameter λ, and any $z_1, z_2 \in \mathcal{I}$, we want that correctness w.r.t. extended evaluation still holds when the memory values are simulated, i.e.*

$$\Pr\left[y = z \cdot P(x_1, \ldots, x_m) : \begin{array}{r} (\mathsf{pk}, (\mathsf{ek}^A, \mathsf{ek}^B)) \leftarrow \mathsf{Setup}(1^\lambda) \\ ([\![x_i]\!]^A, [\![x_i]\!]^B) \leftarrow \mathsf{Input}(\mathsf{pk}, x_i) \forall i \in [m] \\ M^A \leftarrow \mathsf{Sim}_0(1^\lambda) \\ M^B \leftarrow \mathsf{Sim}_1(M^A, z, (\mathsf{ek}^A, \mathsf{ek}^B)) \\ \langle y \rangle^A \leftarrow \mathsf{ExtEval}(A, \mathsf{ek}^A, M^A, ([\![x_i]\!]^A)_{i \in [m]}, P) \\ \langle y \rangle^B \leftarrow \mathsf{ExtEval}(B, \mathsf{ek}^B, M^B, ([\![x_i]\!]^B)_{i \in [m]}, P) \\ y \leftarrow \langle y \rangle^A - \langle y \rangle^B \end{array} \right] \geq 1 - \mathsf{negl}(\lambda).$$

Simulation Security. *There exists an efficient simulator $\mathcal{S}$ such that for all $(\mathsf{pk}, (\mathsf{ek}^A, \mathsf{ek}^B))$ in the image of* Setup*,*

$$\left\{ M_1 \;\middle|\; \begin{array}{r} M_0 \leftarrow \mathsf{Sim}_0(1^\lambda) \\ M_1 \leftarrow \mathsf{Sim}_1(M_0, z, (\mathsf{ek}^A, \mathsf{ek}^B)) \end{array} \right\} \approx_c \mathcal{S}(\mathsf{pk}).$$

3.2 Function Secret Sharing and Distributed Point Functions

Function secret sharing (FSS) [24, 26] is an analogue of HSS for sharing functions instead of data. Concretely, FSS allows to split up a function f into shares $\mathsf{k}_0, \mathsf{k}_1$, such that each share individually hides the function f. The shares can be locally evaluated on an input x, such that the evaluations of both shares at the same x form shares of $f(x)$.

Definition 4 (Function Secret Sharing). *Let $\mathcal{F} = \{f : I \to \mathbb{G}\}$ be a class of function descriptions, where the description of each f specifies the input domain I and an Abelian group $(\mathbb{G}, +)$ as the output domain. A (2-party)* function secret sharing *(FSS) scheme for $\mathcal{F}$ is a pair of algorithms $\mathsf{FSS} = (\mathsf{FSS.Gen}, \mathsf{FSS.Eval})$ with the following syntax:*

- $\mathsf{FSS.Gen}(1^\lambda, f)$ *is a PPT algorithm that given security parameter λ and description of $f \in \mathcal{F}$ outputs a pair of keys (K_0, K_1). We assume that the keys specify I and $\mathbb{G}$.*
- $\mathsf{FSS.Eval}(b, K_b, x)$ *is a polynomial-time algorithm that, given a key K_b for party $b \in \{0, 1\}$, and an input $x \in I$, outputs a group element $y_b \in \mathbb{G}$.*

The scheme should satisfy the following requirements:

Correctness. *For any $f \in \mathcal{F}$ and $x \in I$, we have $\Pr[(K_0, K_1) \xleftarrow{\$} \mathsf{FSS.Gen}(1^\lambda, f) : \mathsf{FSS.Eval}(0, K_0, x) - \mathsf{FSS.Eval}(1, K_1, x) = f(x)] = 1$.*

Security. *Let $f_1, f_2, \ldots$ be any polynomial-size sequence of functions $f_i \in \mathcal{F}$. For $b \in \{0, 1\}$, consider the following two experiments with a PPT simulator* Sim*:*

- $\mathsf{Real}_b(1^\lambda)$*: Run $(\mathsf{k}_0, \mathsf{k}_1) \xleftarrow{\$} \mathsf{FSS.Gen}(1^\lambda, f_\lambda)$ and output k_b*
- $\mathsf{Ideal}_b(1^\lambda)$*: Output $\mathsf{k}_b = \mathsf{Sim}(1^\lambda, \mathsf{Leak}(f_\lambda))$*

We say that FSS *is secure if for each $b \in \{0, 1\}$, there exists* Sim *such that, for any non-uniform adversary $\mathcal{A}$ of size $\mathsf{poly}(\lambda)$, $\Pr[\mathcal{A}(\mathsf{Real}_b(1^\lambda)) = 1] - \Pr[\mathcal{A}(\mathsf{Ideal}_b(1^\lambda)) = 1] \leq \mathsf{negl}(\lambda)$. In the constructions we use, the leakage function $\mathsf{Leak} : \{0, 1\}^* \to \{0, 1\}^*$ is given by $\mathsf{Leak}(f_\lambda) = (I, \mathbb{G})$, namely it outputs a description of the input and output domains of f.*

Remark 5 (Pseudorandomness of FSS Outputs). In some of our constructions, we rely on a property that outputs of FSS.Eval are indistinguishable from random, when the FSS keys remain secret. This property can be obtained, generically, by including a PRF key in each FSS key and having each party add the same PRF evaluation to its output. However, we note that this is already achieved by all the FSS constructions we use.

Distributed Point Functions and FSS for Sums of Point Functions. Let N be a domain size, $\mathbb{G}$ an abelian group, $\alpha \in [N]$, and $\beta \in \mathbb{G}$. A *point function* is a function $f_{\alpha,\beta} : [N] \to \mathbb{G}$ that evaluates to 0 on all inputs $x \neq \alpha$, and $f_{\alpha,\beta}(x) = \beta$ if $x = \alpha$.

An FSS scheme for the class of point functions of domain size N is known as a *distributed point function* [54]. As in previous work [14, 21], it will be convenient for our constructions to use FSS for the class of functions that can be described as a sum of t point functions, which we define next.

Definition 6 (FSS for sum of point functions (SPFSS)). *Let N be a domain size and $\mathbb{G}$ an abelian group. For $S = (s_1, \ldots, s_t) \in [N]^t$ and $\mathbf{y} = (y_1, \ldots, y_t) \in \mathbb{G}^t$, the* sum of point functions *$f_{S,\mathbf{y}} : [N] \to \mathbb{G}$ is defined as*

$$f_{S,\mathbf{y}}(x) = \sum_{i=1}^{t} f_{s_i,y_i}(x).$$

An SPFSS *scheme is an FSS scheme for the class of sums of point functions.*

3.3 Pseudorandom Correlation Generators

A pseudorandom correlation generator, or PCG, is a way of distributing short seeds to a set of parties, which can later be expanded to produce a large amount of correlated randomness. The type of correlated randomness obtained is specified as a correlation generator, which typically has the reverse-samplable property below. In the following definitions, we adapt the definition of n-party PCG [19] to the threshold case where up to t parties may be corrupted. For a set $T \subset [n]$, we denote by $\overline{T}$ the set $[n] \setminus T$.

Definition 7 (Reverse-sampleable t-private n-party Correlation Generator). *A PPT algorithm $\mathcal{C}$ is called a* reverse-sampleable t-private n-party correlation generator*, if on input 1^λ it outputs elements in $\{0,1\}^\ell \times \cdots \times \{0,1\}^\ell \in (\{0,1\}^\ell)^n$ for some length parameter $\ell \in \mathsf{poly}(\lambda)$, and there exists a PPT algorithm $\mathsf{RSample}$ such that for any $T \subset [n]$ of size $\leq t$, the correlation obtained via:*

$$\left\{ (R'_1, \ldots, R'_n) \;\middle|\; \begin{array}{r} (R_1, \ldots, R_n) \xleftarrow{\$} \mathcal{C}(1^\lambda) \\ (R'_\sigma)_{\sigma \in T} := (R_\sigma)_{\sigma \in T} \\ (R'_j)_{j \in \overline{T}} \xleftarrow{\$} \mathsf{RSample}(T, (R_\sigma)_{\sigma \in T}) \end{array} \right\}$$

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

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

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

The algorithms $(\mathsf{PCG.Gen}, \mathsf{PCG.Expand})$ should satisfy the following:

Correctness. *The correlation obtained via:*

$$\{(R_1, \ldots, R_n) | (\mathsf{k}_1, \ldots, \mathsf{k}_n) \xleftarrow{\$} \mathsf{PCG.Gen}(1^\lambda), R_\sigma \leftarrow \mathsf{PCG.Expand}(\sigma, \mathsf{k}_\sigma) \text{ for } \sigma \in [n]\}$$

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

t-Security. *For any $T \subset [n]$, with $|T| \leq t$, the following two distributions are computationally indistinguishable:*

$$\left\{ ((\mathsf{k}_i)_{i \in T}, (R_j)_{j \in [n] \setminus T}) \;\middle|\; \begin{array}{r} (\mathsf{k}_1, \ldots, \mathsf{k}_n) \xleftarrow{\$} \mathsf{PCG.Gen}(1^\lambda) \\ R_j \leftarrow \mathsf{PCG.Expand}(j, \mathsf{k}_j) \;:\; j \in \overline{T} \end{array} \right\}$$

and

$$\left\{ \left((\mathsf{k}_i)_{i\in T}, (R_j)_{j\in[n]\setminus T} \right) \;\middle|\; \begin{array}{r} (\mathsf{k}_1,\ldots,\mathsf{k}_n) \xleftarrow{\$} \mathsf{PCG.Gen}(1^\lambda) \\ R_i \leftarrow \mathsf{PCG.Expand}(i,\mathsf{k}_i) \;:\; i\in T \\ (R_j)_{j\in\overline{T}} \xleftarrow{\$} \mathsf{RSample}(T,(R_i)_{i\in T}) \end{array} \right\},$$

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

3.4 Pseudorandom Correlation Functions

A PCF [20] can be seen as a PCG where each output of the target correlation can be computed "on-demand", instead of being expanded all at once. Furthermore, as in a PRF, the number of possible outputs (that is, the domain size) may be exponential in the security parameter.

The syntax of a PCF consists of a key generation algorithm PCF.KeyGen, which outputs a set of correlated keys, as well as a PCF.Eval algorithm that uses a key to compute a single PCF output. In Appendix A, we give the formal correctness and security properties of a PCF.

3.5 Linear-Output Correlations

Throughout this work, the outputs of our PCGs, PCFs, and HSS schemes are encoded under a *fixed linear secret-sharing scheme* over a finite field, rather than under additive sharing. Extending the "linear HSS" terminology of Fosli et al. [51] (which additionally requires linear *input* sharing), we refer to such primitives as *linear-output* PCGs, PCFs, and HSS schemes, and to the corresponding target correlations as *linear-output correlations*. We first recall the standard matrix-based formulation of linear secret sharing.

Definition 9 (Linear secret-sharing scheme). *A* linear secret-sharing scheme *(LSSS) over a finite field* $\mathbb{F}$ *for* n *parties, with secret space* $\mathbb{F}^k$*, is a tuple* $\Sigma = (M,\psi)$*, where* $M \in \mathbb{F}^{L\times(k+r)}$ *is a* share matrix *and* $\psi\colon [L]\to[n]$ *is a surjective* row-labeling*. To share a secret* $\mathbf{s}\in\mathbb{F}^k$*, sample* $\rho \xleftarrow{R} \mathbb{F}^r$*, compute the* share vector $\mathbf{w} := M\cdot(\mathbf{s}\,\|\,\rho)\in\mathbb{F}^L$*, and output to each party* $i\in[n]$ *the share* $\mathbf{w}[\psi^{-1}(i)]$*, i.e., the subvector of symbols of* $\mathbf{w}$ *labeled with* i*. A set* $T\subseteq[n]$ *of parties is a* privacy set *if the joint distribution of the shares of the parties in* T *is the same for every secret, and a* reconstruction set *if these shares determine the secret. We call* Σ *a* t-private threshold LSSS *if every set of at most* t *parties is a privacy set and every set of at least* $t+1$ *parties is a reconstruction set.*

Equivalently, the set of possible share vectors forms a linear code of length L over $\mathbb{F}$, of which each party receives a fixed subset of symbols. This captures in particular additive sharing ($L=n$, $k=1$, $\psi=\mathrm{id}$, and $\mathbf{w}=(\rho_1,\ldots,\rho_{n-1},\, s-\sum_{j<n}\rho_j)$), degree-$t$ Shamir sharing (the rows of M evaluate the polynomial $s+\rho_1X+\cdots+\rho_tX^t$ at the points $1,\ldots,n$), replicated secret sharing (each additive summand is a symbol labeled, with repetitions, to all parties outside the corresponding subset), and the composed threshold sharing of Section 5.5.

Definition 10 (Linear-output correlation generator). *Let* $\Sigma=(M,\psi)$ *be an LSSS over* $\mathbb{F}$ *for* n *parties with secret space* $\mathbb{F}^k$*, and let* $\mathcal{Y}=\{\mathcal{Y}_\lambda\}_{\lambda\in\mathbb{N}}$ *be an efficiently samplable family of distributions over* $\mathbb{F}^k$ *(the* target distribution*). The* linear-output correlation generator $\mathcal{C}_{\mathcal{Y},\Sigma}$ *is the* n*-party correlation generator that, on input* 1^λ*, samples* $\mathbf{y}\leftarrow\mathcal{Y}_\lambda$ *and* $\rho\xleftarrow{R}\mathbb{F}^r$*, computes* $\mathbf{w}:=M\cdot(\mathbf{y}\,\|\,\rho)$*, and outputs* $R_i:=\mathbf{w}[\psi^{-1}(i)]$ *to each party* $i\in[n]$*.*

Lemma 11. *Let* Σ *be a* t*-private threshold LSSS over* $\mathbb{F}$ *and let* $\mathcal{Y}$ *be efficiently samplable. Then* $\mathcal{C}_{\mathcal{Y},\Sigma}$ *is a reverse-sampleable* t*-private* n*-party correlation generator (Definition 7).*

Proof. Fix $T\subseteq[n]$ with $|T|\le t$. On input the shares $(R_\sigma)_{\sigma\in T}$, the algorithm RSample samples a fresh $\mathbf{y}\leftarrow\mathcal{Y}_\lambda$, then samples ρ uniformly from the affine subspace $\{\rho\in\mathbb{F}^r : (M\cdot(\mathbf{y}\,\|\,\rho))[\psi^{-1}(\sigma)] = R_\sigma \text{ for all } \sigma\in T\}$ (which amounts to solving a linear system over $\mathbb{F}$), and outputs the shares $\mathbf{w}[\psi^{-1}(j)]$, $j\in[n]\setminus T$, of $\mathbf{w}:=M\cdot(\mathbf{y}\,\|\,\rho)$. Since T is a privacy set, the shares $(R_\sigma)_{\sigma\in T}$ are distributed independently of the secret, so the resulting joint distribution is identical to that of $\mathcal{C}_{\mathcal{Y},\Sigma}(1^\lambda)$. ∎

Remark 12 (The correlations in this work). All target correlations considered in this work are linear-output correlations:

- *Threshold-shared correlations* (Section 5): $\mathcal{Y}$ samples $(\mathbf{r}_1, \ldots, \mathbf{r}_d, \otimes_{j=1}^{d} \mathbf{r}_j)$ for uniform $\mathbf{r}_j$, and Σ is degree-t Shamir sharing (or the composed threshold sharing of Section 5.5).
- *Degree-1 Shamir-shared RMS correlations* (Section 4): $\mathcal{Y}$ samples the output of an RMS program on pseudorandom inputs, and Σ is degree-1 Shamir sharing.
- *Three-party permutation correlations*: take $\mathbb{F} = \mathbb{F}_3$, let $\mathcal{Y}$ be a distribution over $\mathbb{F}_3^* = \{1, 2\}$ (uniform for the unbiased correlation of Section 4, biased for the construction of Appendix B), and let Σ share $\nu \in \mathbb{F}_3^*$ via $\mathbf{w} = (x,\ x + \nu,\ x - \nu)$ for uniform $x \in \mathbb{F}_3$, i.e., a degree-1 Shamir sharing of ν *in the leading coefficient*, evaluated at the points $0, 1, -1$. Since $\nu \neq 0$ and $\mathbb{F}_3$ has characteristic 3, the three symbols $x, x + \nu, x - \nu$ are pairwise distinct, so the output triple is a (pseudo)random permutation of $\mathbb{F}_3$. (This is the substitution $\nu := 2 - \mu$ in the encoding of Section 2.2.) The difference with an unconstrained linear-output correlation is only the support restriction on the target distribution $\mathcal{Y}$.

Linear correlations [11, 53] correspond to the special case where $\mathcal{Y}$ is the uniform distribution over $\mathbb{F}^k$ (the joint output is then a fixed linear function of uniform randomness), which is exactly the case handled by pseudorandom secret sharing. The focus of this work is on linear-output correlations whose target distribution is *structured*: sampled by a low-degree or RMS/NC^1 computation over pseudorandom inputs, or supported on a strict subset of the secret space (as for permutations).

4 PCF for Permutations

In this section, we describe a construction that is based on two-party HSS with simulatable memory shares to obtain a PCF for unbiased, 3-party permutation correlations. The construction follows the overview from Section 2.2, by building an HSS with degree-1 Shamir-shared outputs and using this to evaluate a pseudorandom function. We start by directly presenting the 3-party PPF construction, and then later describe how the same ideas can be used to build an n-party HSS construction with degree-1 Shamir-shared outputs.

For this, let P be an HSS program for a PRF with 1-bit output. Denote it by F_κ, where κ is the PRF key. To run it, we need HSS shares of the key, generated via $(\llbracket\kappa\rrbracket^A, \llbracket\kappa\rrbracket^B) \leftarrow \mathsf{HSS.Input}(\mathsf{pk}, \kappa)$. Then, Alice gets $\llbracket\kappa\rrbracket^A$, and both, Bob and Carol get $\llbracket\kappa\rrbracket^B$.

Now, when Alice and either Bob or Carol run $\mathsf{Eval}(\sigma, \mathsf{ek}^\sigma, \llbracket\kappa\rrbracket^\sigma, P_x)$, they obtain, upon reconstruction, a pseudorandom bit $\mu = F.\mathsf{Eval}(\kappa, x)$. Recall that we want Alice to obtain a share that, together with the share of Bob, is a share of a pseudorandom bit μ, and, together with the share of Carol, is a share of $1 - \mu$. Hence, we want to compute $\alpha \cdot \mu$, where in one case, $\alpha = 1$, and in the other case, $\alpha = -1$. To wit, it suffices that Carol subtracts 1 from her subtractive share, thereby converting it to a share of $1 - \mu$.

To do this, we can provide the α as memory shares and evaluate P_x for a PRF input x, using our extended evaluation (cf. Section 3.1). And, as we want to keep Alice's share fixed, we do so by using memory shares M^A, M^B that are simulatable, i.e., Alice uses $M^A \leftarrow \mathsf{Sim}_0(1^\lambda)$ and to get a memory share, whose value is not yet fixed, but can be determined by generating a share using Sim_1.

Later, this is programmed twice, to encode certain $\alpha_B \in \mathcal{I}$ but also $\alpha_C \in \mathcal{I}$ using Sim_1, for use with Bob and Carol, respectively. Then, $\mathsf{ExtEval}$ will output a share of $\alpha_p \cdot F.\mathsf{Eval}(\kappa, x)$ for $p \in \{B, C\}$, where $F.\mathsf{Eval}(\kappa, x)$ is a pseudorandom bit as output by the PRF. As motivated above, we choose $\alpha_B = 1$ and $\alpha_C = -1$. Although our construction internally programs pairwise subtractive shares, it realizes the linear-output permutation correlation of Remark 12 — a degree-1 sharing over $\mathbb{F}_3$ whose target distribution is supported on $\mathbb{F}_3^*$, followed by fixed local transformations. The additional point needed for the permutation correlation is that the shared coefficient is restricted to $\{0, 1\}$, which guarantees that the resulting three outputs are pairwise distinct. See the general template in Section 2.2 for the pairwise-distinctness argument.

We write $\mathcal{C}_{\mathsf{perm}}$ for the three-party permutation correlation that samples $\pi \xleftarrow{\mathrm{R}} S_3$ uniformly and outputs $\pi(0), \pi(1), \pi(2)$ to the three parties, respectively. For simplicity of notation, we identify S_3 as the set of permutations over the elements of $\mathbb{F}_3$.

Theorem 13. *Let* $\mathsf{HSS} = (\mathsf{Setup}, \mathsf{Input}, \mathsf{Eval})$ *be an HSS with simulatable memory shares (with algorithms* $\mathsf{Sim}_0, \mathsf{Sim}_1$*) and with input space* $\mathcal{I} = \mathbb{F}_3$ *and share space* $\mathcal{S} = \mathbb{F}_3$*. Moreover, let* F *be a pseudorandom function and for* $x \in \{0,1\}^k$*, let* P_x *be the HSS program that runs* $F.\mathsf{Eval}(\cdot, x)$ *when given the PRF key as input. Then, the PCF of Figure 1 is a PCF for* $\mathcal{C}_{\mathsf{perm}}$*.*

Proof. We show each property in turn.

Correctness. Let $(\mathsf{k}_0, \mathsf{k}_1, \mathsf{k}_2)$ as output by $\mathsf{PCF.Gen}(1^\lambda)$. Let k_i be parsed as $(\mathsf{pk}, [\![\kappa]\!]_i, \mathsf{ek}_i, M^i)$ for $i \in \{0,1,2\}$. Moreover, let $R_i \leftarrow \mathsf{PCF.Eval}(i, \mathsf{k}_i, x)$, for some $x \in \{0,1\}^k$. More concretely,

$$
\begin{aligned}
R_0 &= \mathsf{ExtEval}(A, \mathsf{ek}^A, M^A, [\![\kappa]\!]^A, P_x) \\
R'_1 &= \mathsf{ExtEval}(B, \mathsf{ek}^B, M^B, [\![\kappa]\!]^B, P_x) \\
R'_2 &= \mathsf{ExtEval}(B, \mathsf{ek}^B, M^C, [\![\kappa]\!]^B, P_x) - 1 \\
R_1 &= 2 + R'_1 \\
R_2 &= 2 + R'_2.
\end{aligned}
$$

We need to show that $(R_0, R_1, R_2) \in \mathbb{F}_3^3$ as obtained in this manner, is a uniformly random permutation over the elements of $\mathbb{F}_3$.

By the simulation correctness of the extended evaluation, we know that by construction, with overwhelming probability, $R_0 - R'_1 = F.\mathsf{Eval}(\kappa, x) \in \{0,1\}$, and $R_0 - R'_2 = 1 - F.\mathsf{Eval}(\kappa, x) \in \{0,1\}$. Also, because the distinguisher only gets the 3-tuple, by the security of the HSS scheme, they cannot learn anything about κ, and hence, $F.\mathsf{Eval}(\kappa, x)$ is computationally indistinguishable from a random bit. Then, using $R'_1 = R_0 - \mu$, and $R'_2 = R_0 + \mu - 1$, for $\mu := F.\mathsf{Eval}(\kappa, x)$, we get $R_0 + R_1 + R_2 = R_0 + (2 + R'_1) + (2 + R'_2) = R_0 + 2 + (R_0 - \mu) + (R_0 + 1 + \mu) = 3R_0 + 2 - \mu + 1 + \mu = 0 \pmod 3$, i.e., the R's form an additive sharing of zero, as required.

Next, we show that they are pairwise distinct. This is because $R_0 = R_1$ implies $R_0 - R'_1 = \mu = 2$, which is false. The other cases are implied and/or similar. Hence, (R_0, R_1, R_2) forms a permutation. It remains to show that it is uniformly generated. As R_0 is a fresh random subtractive share, it follows if the HSS satisfies the additional property that the marginal distribution of an output share is computationally close to random. While this does not follow automatically from the HSS definition, it is folklore that any HSS can be trivially modified to satisfy this slightly stronger property by letting $\mathsf{HSS.Input}$ additionally output a fresh PRF key (for a PRF with range $\mathbb{F}_3$) in both input shares, and letting $\mathsf{HSS.Eval}$ sum the PRF output (evaluated on the RMS program viewed as a string) to its output. This preserves correctness (the PRF evaluations cancel out when subtracting the shares) and guarantees that each share is individually pseudorandom. With this change, R_0 is computationally close to uniform over $\mathbb{F}_3$. Then, given this R_0, we show that the two possibilities for R_1 are exactly determined by the pseudorandom bit of the PRF, so it is indistinguishable from being uniformly chosen from $\mathbb{F}_3 \setminus \{R_0\}$.

Security. Here, we show PCF security for each $\sigma \in \{0,1,2\}$. Note that the proof is different for the different σ's, because for $\sigma = 0$, the memory share M^A that is part of the key does not need to be simulated. We prove the case where $\sigma = 0$. The other cases follow via an analogous proof.

Hybrid $\mathcal{H}_0$*.* This hybrid consists of $\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,\sigma,1}(\lambda)$ game as defined in Figure 4.

Hybrid $\mathcal{H}_1$*.* In this hybrid, we change how the output $R_1^{(i)}$ of Bob is computed (for all $i \in [N]$). First, we reconstruct the PRF key κ from the HSS input shares contained in the PCF keys and then evaluate the PRF on input x_i to get the output μ_i. Then, we use Alice's output $R_0^{(i)}$ to compute Bob's share $R_1^{(i)}$ as $R_0^{(i)} + 2 - \mu_i$.

We note that $\mathcal{H}_1$ is identical to $\mathcal{H}_0$, since the change is merely syntactic. Now, in $\mathcal{H}_1$, Bob's share is no longer computed using k_1.

Hybrid $\mathcal{H}_2$*.* In this hybrid, we change how the output of Carol is computed. As before, we first reconstruct the PRF key κ from the HSS input shares contained in the PCF keys and then evaluate the PRF on input x_i to get the output μ_i. Then, we use Alice's output $R_0^{(i)}$ to compute Carol's share $R_2^{(i)}$ as $R_0^{(i)} + 1 + \mu_i$.

We note that $\mathcal{H}_2$ is again identical to $\mathcal{H}_1$, since the change is merely syntactic. Now, in $\mathcal{H}_2$, Carol's share is no longer computed using k_2.

Hybrid $\mathcal{H}_3$. In this hybrid, we replace the HSS input shares of κ with an input share of 0.

$\mathcal{H}_3 \approx_c \mathcal{H}_2$ by the security of HSS.

Hybrid $\mathcal{H}_4$. In this hybrid, we replace the pseudorandom bit μ_i (as output in $\mathcal{H}_3$ using the PRF) with a uniformly random bit.

$\mathcal{H}_4 \approx_c \mathcal{H}_3$ by the pseudorandomness of the PRF.

Hybrid $\mathcal{H}_5$. This hybrid consists of $\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,\sigma,0}(\lambda)$ game as defined in Figure 4.

$\mathcal{H}_5$ is identical to $\mathcal{H}_4$ given that in $\mathcal{H}_4$, we already sample $R_1^{(i)}$ and $R_2^{(i)}$ entirely from Alice's output $R_0^{(i)}$.

The argument above proves security for a singleton corrupted set T. For $|T| = 2$, the two corrupted parties can compute their own two permutation outputs from their keys, and these outputs uniquely determine the remaining output. Thus the reverse-sampling algorithm has no choice for the honest party's output, and the real and ideal games coincide except with the negligible correctness error of the construction. For $|T| = 3$, there are no honest outputs to simulate, so the two security games are identical. Hence the construction satisfies the PCF security definition for every corrupted set $T \subseteq \{0, 1, 2\}$; the one-corruption case is the strongest case providing non-trivial privacy about the remaining permutation.

■

HSS-based PCF for pseudorandom permutations over $\mathbb{F}_3$

$\mathsf{Gen}(1^\lambda)$

1 : $(\mathsf{pk}, (\mathsf{ek}^A, \mathsf{ek}^B)) \leftarrow \mathsf{HSS.Setup}(1^\lambda)$
2 : $\kappa \leftarrow F.\mathsf{KeyGen}(1^\lambda)$
3 : $([\![\kappa]\!]^A, [\![\kappa]\!]^B) \leftarrow \mathsf{HSS.Input}(\mathsf{pk}, \kappa)$
4 : $M^A \leftarrow \mathsf{Sim}_0(1^\lambda)$
5 : $M^B \leftarrow \mathsf{Sim}_1(M^A, \alpha_B := 1, (\mathsf{ek}^A, \mathsf{ek}^B))$
6 : $M^C \leftarrow \mathsf{Sim}_1(M^A, \alpha_C := -1, (\mathsf{ek}^A, \mathsf{ek}^B))$
7 : Set $\mathsf{k}_0 := (\mathsf{pk}, [\![\kappa]\!]^A, \mathsf{ek}^A, M^A)$
8 : Set $\mathsf{k}_1 := (\mathsf{pk}, [\![\kappa]\!]^B, \mathsf{ek}^B, M^B)$
9 : Set $\mathsf{k}_2 := (\mathsf{pk}, [\![\kappa]\!]^B, \mathsf{ek}^B, M^C)$
10 : **return** $(\mathsf{k}_0, \mathsf{k}_1, \mathsf{k}_2)$

$\mathsf{Eval}(i, \mathsf{k}_i, x)$

1 : **parse** $\mathsf{k}_i = (\mathsf{pk}, k', \mathsf{ek}, M)$
2 : $\rho \leftarrow A$ if $i = 0$ and B otherwise
3 : $\langle \mu \rangle_i \leftarrow \mathsf{ExtEval}(\rho, \mathsf{ek}, M, k', P_x)$ ▷ P_x is the program for $F.\mathsf{Eval}(\cdot, x)$
4 : **if** $i = 0$: **return** $\langle \mu \rangle_i$
5 : **if** $i = 1$: **return** $2 + \langle \mu \rangle_i$
6 : **else return** $1 + \langle \mu \rangle_i$

Fig. 1: HSS-based PCF construction

Concrete Performance. Although this construction makes non-black-box use of a PRF, with some care, it can still be reasonably efficient for practical applications. In Appendix B.7, we show that by combining HSS schemes with a suitable, low-complexity weak PRF, we can plausibly obtain constructions that allow expanding short seeds to produce up to 2^{40} pseudorandom permutation correlations, where the PPF for one party's output can be evaluated in under 100ms.

Extension to degree-1 Shamir-output HSS. As mentioned in Section 2.2, we can use the same idea as in our PPF construction to build an n-party HSS scheme with degree-1 Shamir-shared outputs, secure against one corruption. The construction supports evaluation of the same class of functions as the underlying two-party HSS, so when instantiating this under LWE, DCR, or class-group assumptions, we obtain an n-party HSS with degree-1 Shamir-shared outputs for RMS programs (including all of NC^1).

The construction is shown in Figure 2. We associate party P_i, for $i \in \{0, \ldots, n-1\}$, with the standard Shamir evaluation point

$$\xi_i := i + 1 \in \mathbb{F}_p.$$

For each $i \geq 1$, we use the simulatable-memory property to program

$$\alpha_i := \frac{i}{i+1} \in \mathbb{F}_p.$$

If y_A and $y_B^{(i)}$ are the corresponding outputs of the underlying two-party extended evaluation, then

$$y_A - y_B^{(i)} = \alpha_i P(x_1, \ldots, x_m).$$

Party P_0 outputs y_A, while party P_i, for $i \geq 1$, outputs $(i+1)y_B^{(i)}$. As shown below, these values are evaluations at $\xi_0, \ldots, \xi_{n-1}$ of a single degree-1 polynomial whose value at zero is $P(x_1, \ldots, x_m)$.

Theorem 14. *Let*

$$\mathsf{HSS}_2 = (\mathsf{HSS}_2.\mathsf{Setup}, \mathsf{HSS}_2.\mathsf{Input}, \mathsf{HSS}_2.\mathsf{Eval})$$

be a two-party HSS with simulatable memory shares, with algorithms $\mathsf{HSS}_2.\mathsf{Sim}_0, \mathsf{HSS}_2.\mathsf{Sim}_1$*, and share space* $\mathcal{S} = \mathbb{F}_p$ *for some prime* $p > n$*. Assume that the memory-value domain contains* $\alpha_i = i/(i+1)$ *for every* $i \in \{1, \ldots, n-1\}$*. Then the scheme* HSS *of Figure 2 is an* n*-party HSS scheme secure against one corruption, supporting the same class of programs as* HSS_2*, whose evaluated outputs lie on a degree-*1 *Shamir polynomial with secret* $P(x_1, \ldots, x_m)$ *at zero.*

Moreover, whenever the first output share of the underlying extended evaluation is computationally indistinguishable from uniform over $\mathbb{F}_p$*, the resulting output sharing is computationally indistinguishable from a standard random degree-*1 *Shamir sharing of* $P(x_1, \ldots, x_m)$*.*

Proof. We prove correctness and then argue security.

Correctness. Let

$$P^\star := P(x_1, \ldots, x_m).$$

For party P_0, define

$$y_A := \mathsf{HSS}_2.\mathsf{ExtEval}\big(A, \mathsf{ek}^A, M^0, (\llbracket y_j \rrbracket^A)_{j=1}^m, P\big).$$

For each $i \in \{1, \ldots, n-1\}$, let

$$y_B^{(i)} := \mathsf{HSS}_2.\mathsf{ExtEval}\big(B, \mathsf{ek}^B, M^i, (\llbracket y_j \rrbracket^B)_{j=1}^m, P\big).$$

By correctness of extended evaluation and the way M^i is programmed, except with negligible probability,

$$y_A - y_B^{(i)} = \alpha_i P^\star = \frac{i}{i+1} P^\star.$$

Thus

$$y_B^{(i)} = y_A - \frac{i}{i+1} P^\star.$$

The output of party P_0 is

$$[\![y]\!]^0 = y_A,$$

while the output of party P_i, for $i \geq 1$, is

$$[\![y]\!]^i = (i+1)y_B^{(i)} = (i+1)y_A - iP^\star.$$

Consider the degree-1 polynomial

$$q(z) := P^\star + z\big(y_A - P^\star\big).$$

Its constant term is

$$q(0) = P^\star.$$

Moreover, recalling that party P_i is associated with the Shamir evaluation point $\xi_i = i+1$, we have

$$q(\xi_0) = q(1) = y_A = [\![y]\!]^0,$$

and, for every $i \geq 1$,

$$\begin{aligned} q(\xi_i) &= q(i+1) \\ &= P^\star + (i+1)(y_A - P^\star) \\ &= (i+1)y_A - iP^\star \\ &= (i+1)y_B^{(i)} \\ &= [\![y]\!]^i. \end{aligned}$$

Hence the parties' outputs are evaluations of a single degree-1 polynomial at the distinct nonzero points $1, \ldots, n$, and this polynomial has secret $P^\star$ at zero. Consequently any two output shares reconstruct $P^\star$.

If y_A is computationally indistinguishable from uniform over $\mathbb{F}_p$, then so is the coefficient $y_A - P^\star$. Therefore the resulting vector

$$\big(q(1), \ldots, q(n)\big)$$

is computationally indistinguishable from a standard random degree-1 Shamir sharing of $P^\star$.

Security. Consider a corrupted party P_i.

If $i = 0$, its view consists of the public key, the A-side evaluation key and input shares of the underlying two-party HSS, together with the memory value M^0 generated by $\mathsf{HSS}_2.\mathsf{Sim}_0$. Security therefore follows from the security of the underlying HSS and the simulatable-memory property.

If $i \geq 1$, its view consists of the corresponding B-side evaluation key and input shares together with

$$M^i \leftarrow \mathsf{HSS}_2.\mathsf{Sim}_1\left(M^0, \frac{i}{i+1}, (\mathsf{ek}^A, \mathsf{ek}^B)\right).$$

By the simulatable-memory property, this memory value can be simulated without revealing the underlying HSS input. The additional operation performed by party P_i is multiplication of its extended-evaluation output by the public, nonzero scalar $i+1$, which is an invertible local transformation and reveals no additional information. Thus security against one corrupted party follows from the security of HSS_2 and its simulatable-memory property. ■

5 Multiparty PCG for Correlations with Threshold Secret Sharings

In this section, we construct multiparty PCGs for constant-degree correlations with threshold secret sharings from multiparty DPF with additive sharings.

Notations. $\binom{[n]}{t}$ denotes the collection of all subsets in $[n]$ of size t. $\binom{[n]}{\leq t}$ denotes the collection of all subsets in $[n]$ of size at most t. $\binom{[n]}{\geq t}$ denotes the collection of all subsets in $[n]$ of size at least t. As in Section 3.1, t denotes the privacy threshold. To avoid ambiguity with reconstruction-threshold terminology, we write $\mathsf{DPF}_{n,t}$, $\mathsf{PCG}_{n,t}$, and $\mathsf{HSS}_{n,t}$ for n-party primitives secure against up to t corruptions.

HSS Scheme for degree-1 Shamir-shared outputs

$\mathsf{Setup}(1^\lambda)$

1 : $(\mathsf{pk}, (\mathsf{ek}^A, \mathsf{ek}^B)) \leftarrow \mathsf{HSS}_2.\mathsf{Setup}(1^\lambda)$
2 : $M^0 \leftarrow \mathsf{HSS}_2.\mathsf{Sim}_0(1^\lambda)$
3 : **foreach** $i \in \{1, \ldots, n-1\}$:
4 : $\quad \alpha_i := i \cdot (i+1)^{-1} \in \mathbb{F}_p$
5 : $\quad M^i \leftarrow \mathsf{HSS}_2.\mathsf{Sim}_1\big(M^0, \alpha_i, (\mathsf{ek}^A, \mathsf{ek}^B)\big)$
6 : $\mathsf{ek}_0 := (\mathsf{pk}, \mathsf{ek}^A, M^0)$
7 : **foreach** $i \in \{1, \ldots, n-1\}$:
8 : $\quad \mathsf{ek}_i := (\mathsf{pk}, \mathsf{ek}^B, M^i)$
9 : **return** $\big(\mathsf{pk}, (\mathsf{ek}_0, \ldots, \mathsf{ek}_{n-1})\big)$

$\mathsf{Input}(\mathsf{pk}, x)$

1 : $(\llbracket x \rrbracket^A, \llbracket x \rrbracket^B) \leftarrow \mathsf{HSS}_2.\mathsf{Input}(\mathsf{pk}, x)$
2 : **return** $(\llbracket x \rrbracket^A, \llbracket x \rrbracket^B, \ldots, \llbracket x \rrbracket^B)$

$\mathsf{Eval}\big(i, \mathsf{ek}_i, (\llbracket y_j \rrbracket^i)_{j=1}^m, P\big)$

1 : **parse** $\mathsf{ek}_i = (\mathsf{pk}, \mathsf{ek}', M^i)$
2 : $\rho \leftarrow A$ if $i = 0$ and B otherwise
3 : $\widetilde{y}_i \leftarrow \mathsf{HSS}_2.\mathsf{ExtEval}\big(\rho, \mathsf{ek}', M^i, (\llbracket y_j \rrbracket^i)_{j=1}^m, P\big)$
4 : **if** $i = 0$: **return** $\widetilde{y}_i$
5 : **else return** $(i+1)\widetilde{y}_i$

$\mathsf{Rec}_T((\llbracket y \rrbracket^i)_{i \in T})$

1 : Choose any distinct $i, j \in T$
2 : $\xi_i := i + 1, \qquad \xi_j := j + 1$
3 : $\lambda_i := \dfrac{-\xi_j}{\xi_i - \xi_j}, \qquad \lambda_j := \dfrac{-\xi_i}{\xi_j - \xi_i}$
4 : **return** $\lambda_i \llbracket y \rrbracket^i + \lambda_j \llbracket y \rrbracket^j$

Fig. 2: n-party HSS with degree-1 Shamir-shared outputs based on a two-party HSS scheme with simulatable memory shares. Party P_i is associated with Shamir evaluation point $\xi_i = i + 1$.

5.1 Shamir secret sharing and reverse sampling

We use the linear secret-sharing terminology of Definition 9. In particular, a degree-t Shamir sharing is a t-private threshold LSSS: to share a secret $s \in \mathbb{F}$, sample a uniformly random degree-t polynomial $p(X) = s + \rho_1 X + \cdots + \rho_t X^t$ over $\mathbb{F}$ and give party P_i the share $s_i := p(i)$. Any t shares reveal no information about s, while any $t+1$ shares reconstruct s by interpolation.

We denote the resulting sharing $(s_1, \ldots, s_n)$ by $[s]_t^n$. For a vector $\mathbf{r} = (r_1, \ldots, r_m)$, we write $[\mathbf{r}]_t := ([r_1]_t, \ldots, [r_m]_t)$. Shamir sharing is linearly homomorphic, so in particular $[x_1]_t + [x_2]_t = [x_1 + x_2]_t$.

The reverse-samplability of Shamir sharing follows from Lemma 11. We will repeatedly use the following explicit specialization. Given a secret $r \in \mathbb{F}$ and the shares $\{r_j\}_{j \in T}$ of any set $T \in \binom{[n]}{\leq t}$, $\mathsf{RandShamir}(r, t, T, \{r_j\}_{j \in T})$ samples a uniformly random degree-t polynomial p conditioned on $p(0) = r$ and $p(j) = r_j \quad$ for every $j \in T$, and outputs $(p(j))_{j \in [n] \setminus T}$.

Concretely, this can be implemented by choosing $m := t - |T|$ parties $P_{i_1}, \ldots, P_{i_m}$ outside T, sampling $p(i_1), \ldots, p(i_m)$ uniformly, and interpolating the unique degree-t polynomial satisfying these values together with $p(0) = r$ and the prescribed shares in T.

5.2 Multiparty DPF, PCG, HSS with threshold secret sharing schemes

We next define the multiparty DPF notion used in our constructions. Its outputs are shares under a fixed t-private threshold LSSS Σ, and security is required against any set of at most t corrupted parties. We use the general t-private n-party PCG notion from the preliminaries and do not repeat it here.

Definition 15 (t-private n-party DPF). *An n-party DPF with output sharing scheme Σ, secure against up to t corruptions, for point functions with input domain $[D]$ and output domain a finite field $\mathbb{F}$, 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 $(k_1, \ldots, k_n)$;*
- $\mathsf{DPF.Eval}(\sigma, k_\sigma, x)$*, given party index $\sigma \in [n]$, key k_σ, and input $x \in [D]$, outputs a field element $y_\sigma \in \mathbb{F}$.*

The scheme DPF should satisfy the following requirements:

- ***Correctness.*** *For any $(\alpha, \beta) \in [D] \times \mathbb{F}$ and $x \in [D]$, we have with probability 1 over the randomness of $\mathsf{DPF.Gen}(1^\lambda, \alpha, \beta)$, letting $y_\sigma \leftarrow \mathsf{DPF.Eval}(\sigma, k_\sigma, x)$ for each $\sigma \in [n]$, $(y_1, \ldots, y_n)$ forms a valid Σ-sharing with secret $f_{\alpha,\beta}(x)$.*
- ***Security.*** *For every set of corrupted parties $T \in \binom{[n]}{\leq t}$, there exists a PPT simulator Sim such that for any family $\mathcal{F} = \{f_{\alpha_\lambda,\beta_\lambda} : [D(\lambda)] \to \mathbb{F}_\lambda\}_{\lambda \in \mathbb{N}}$ of point functions over domain $[D(\lambda)]$ and field $\mathbb{F}_\lambda$, the distributions*

$$\{(k_\sigma)_{\sigma \in T} \mid (k_1, \ldots, k_n) \leftarrow \mathsf{DPF.Gen}(1^\lambda, \alpha_\lambda, \beta_\lambda)\}$$

and

$$\{(k_\sigma)_{\sigma \in T} \mid (k_\sigma)_{\sigma \in T} \leftarrow \mathsf{Sim}(1^\lambda, D(\lambda), \mathbb{F}_\lambda)\}$$

are computationally indistinguishable.

Notice that an n-party DPF with additive sharing is an $(n-1)$-private n-party DPF.

The following property is useful to argue the correctness of a PCG compiled from a DPF.

Definition 16 (t-private n-party DPF with pseudorandom outputs). *We say that a t-private n-party DPF scheme $(\mathsf{DPF.Gen}, \mathsf{DPF.Eval})$ with output sharing scheme Σ for family $\mathcal{F} = \{f_{\alpha,\beta} : [D] \to \mathbb{F}\}$ of point functions satisfies pseudorandomness of outputs, if for all $f \in \mathcal{F}, x \in [D]$, and subset of keys $T \in \binom{[n]}{\leq t}$, the following distributions*

$$\left\{ (k_\sigma)_{\sigma \in T}, (y_{\tilde{\sigma}})_{\tilde{\sigma} \in [n] \setminus T} \;\middle|\; \begin{array}{c} (k_1, \ldots, k_n) \leftarrow \mathsf{DPF.Gen}(1^\lambda, \alpha, \beta) \\ y_{\tilde{\sigma}} \leftarrow \mathsf{DPF.Eval}(\tilde{\sigma}, k_{\tilde{\sigma}}, x) \text{ for } \tilde{\sigma} \in [n] \setminus T \end{array} \right\}$$

and

$$\left\{ (k_\sigma)_{\sigma\in T}, (y_{\tilde\sigma})_{\tilde\sigma\in[n]\setminus T} \;\middle|\; \begin{array}{c} (k_1,\ldots,k_n) \leftarrow \mathsf{DPF.Gen}(1^\lambda,\alpha,\beta) \\ y_\sigma \leftarrow \mathsf{DPF.Eval}(\sigma,k_\sigma,x) \text{ for } \sigma\in T \\ (y_{\tilde\sigma})_{\tilde\sigma\in[n]\setminus T} \leftarrow \mathsf{RandSharing}(f_{\alpha,\beta}(x),t,T,\{y_\sigma\}_{\sigma\in T}) \end{array} \right\}$$

are computationally indistinguishable, where RandSharing *is the reverse sampling algorithm for Σ-sharing scheme.*

For the DPF-to-PCG-to-HSS transformations used below, we also need the following share/evaluate formulation of multiparty HSS and its pseudorandom-output property.

Definition 17 (t-private n-party HSS). *An n-party HSS with output sharing scheme Σ, secure against up to t corruptions, for function family $\mathcal{F}$ 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 $x \in \mathbb{F}$ (the input), the sharing algorithm* Share *outputs n inputs shares $(x^{(1)},\ldots,x^{(n)})$.*
- $\mathsf{Eval}(\sigma, f, x^{(\sigma)})$*: 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 length I, O), and $x^{(\sigma)}$ (the σ-th input share), the evaluation algorithm* Eval *outputs the σ-th output share $y^{(\sigma)} \in \mathbb{F}^O$.*
- ***Correctness****. For any 1^λ, input $x \in \mathbb{F}^{I(\lambda)}$, and any function $f \in \mathcal{F}$, letting*

$$\begin{array}{c} (x^{(1)},\ldots,x^{(n)}) \leftarrow \mathsf{HSS.Share}(1^\lambda, x) \\ y^{(\sigma)} \leftarrow \mathsf{HSS.Eval}(\sigma, f, x^{(\sigma)}) \text{ for } \sigma \in [n], \end{array}$$

  then $(y^{(1)},\ldots,y^{(n)})$ forms valid Σ-sharings of $f(x)$.
- ***Security****. For any set of corrupted parties $T \in \binom{[n]}{\le t}$, there exists a PPT algorithm* Sim*, such that for every sequence of inputs $x_1, x_2, \ldots \in \mathbb{F}^{I(\lambda)}$, the following distributions*

$$\left\{ (x^{(\sigma)})_{\sigma\in T} \;\middle|\; (x^{(1)},\ldots,x^{(n)}) \leftarrow \mathsf{HSS.Share}(1^\lambda, x) \right\}$$

  and

$$\left\{ (x^{(\sigma)})_{\sigma\in T} \;\middle|\; (x^{(\sigma)})_{\sigma\in T} \leftarrow \mathsf{Sim}(1^\lambda, n, I, O) \right\}$$

  are computationally indistinguishable.

Definition 18 (t-private n-party HSS with pseudorandom outputs). *We say that a t-private n-party HSS scheme* $(\mathsf{HSS.Share}, \mathsf{HSS.Eval})$ *with output sharing scheme Σ for function family $\mathcal{F}$ over a finite field $\mathbb{F}$ satisfies pseudorandomness of outputs, if for all $f \in \mathcal{F}, x \in \mathbb{F}^{I(\lambda)}$, and subset of shares $T \in \binom{[n]}{\le t}$, the following distributions*

$$\left\{ (x^{(\sigma)})_{\sigma\in T}, (y^{(\tilde\sigma)})_{\tilde\sigma\in[n]\setminus T} \;\middle|\; \begin{array}{c} (x^{(1)},\ldots,x^{(n)}) \leftarrow \mathsf{HSS.Share}(1^\lambda, x) \\ y^{(\tilde\sigma)} \leftarrow \mathsf{HSS.Eval}(\tilde\sigma, f, x^{(\tilde\sigma)}) \text{ for } \tilde\sigma \in [n]\setminus T \end{array} \right\}$$

and

$$\left\{ (x^{(\sigma)})_{\sigma\in T}, (y_{\tilde\sigma})_{\tilde\sigma\in[n]\setminus T} \;\middle|\; \begin{array}{c} (x^{(1)},\ldots,x^{(n)}) \leftarrow \mathsf{HSS.Share}(1^\lambda, x) \\ y_\sigma \leftarrow \mathsf{HSS.Eval}(\sigma, f, x^{(\sigma)}) \text{ for } \sigma \in T \\ (y_{\tilde\sigma})_{\tilde\sigma\in[n]\setminus T} \leftarrow \mathsf{RandSharing}(f(x),t,T,\{y^{(\sigma)}\}_{\sigma\in T}) \end{array} \right\}$$

are computationally indistinguishable, where RandSharing *is the reverse sampling algorithm for Σ-sharing scheme.*

When the output sharing scheme Σ is linear — as is the case for all schemes constructed in this work (degree-t Shamir sharing and the composed threshold sharing of Section 5.5) — these are *linear-output* DPFs, PCGs, and HSS schemes in the sense of Definition 10.

5.3 t-private n-party PCG for constant-degree polynomials from additive DPF: constant n and t

We first recall local share conversion from replicated secret sharing scheme to Shamir secret sharing scheme [43]. To share a secret $s \in \mathbb{F}$ with replicated secret sharing, we first additively and randomly decompose s into $\binom{n}{t}$ pieces, where each piece s_T is labeled with a subset $T \in \binom{[n]}{t}$, i.e., $s = \sum_{T \in \binom{[n]}{t}} s_T$. Then we assign each s_T to parties that are not in T. Now for each party P_i, it will obtain a set of shares $\{s_T\}_{T \in \binom{[n]}{t}: i \notin T}$.

To transform from replicated secret sharing of s to degree-t Shamir sharing of s, for each subset $T \in \binom{[n]}{t}$, all parties define a degree-t polynomial $f_T : \mathbb{F} \to \mathbb{F}$ such that

$$f_T(0) = 1 \text{ and } \forall i \in T, f_T(i) = 0.$$

Let $f : \mathbb{F} \to \mathbb{F}$ be a degree-t polynomial such that $f = \sum_{T \in \binom{[n]}{t}} s_T \cdot f_T$. Then $f(0) = \sum_{T \in \binom{[n]}{t}} s_T \cdot f_T(0) = \sum_{T \in \binom{[n]}{t}} s_T = s$. Observe that each party P_i can locally compute

$$f(i) = \sum_{T \in \binom{[n]}{t}} s_T \cdot f_T(i) = \sum_{T \in \binom{[n]}{t}: i \notin T} s_T \cdot f_T(i),$$

as $f_T(i) = 0$ when $i \in T$ and s_T is known to P_i for each $T \in \binom{[n]}{t}$ satisfying $i \notin T$.

As for the privacy, for any subset of parties $\mathcal{C} \in \binom{[n]}{<t}$, there exists $T_0 \in \binom{[n]}{t}$ such that $\mathcal{C} \subseteq T_0$. Then s_{T_0} and thus s are unknown to parties in $\mathcal{C}$. As for reconstruction property, considering any subset of parties $\mathcal{I} \in \binom{[n]}{\geq t+1}$, for every $T \in \binom{[n]}{t}$, $\mathcal{I} - T \neq \emptyset$ and thus s_T is known to parties in $\mathcal{I}$. Hence, $s = \sum_{T \in \binom{[n]}{t}} s_T$ can be computed by parties in $\mathcal{I}$.

In short, if we can additively share the secret, redundancy can be added by traversing all access structures. Based on this recipe, instead of holding a replicated secret sharing of a single secret s, all parties hold replicated sharing of a point function and are expected to compute their degree-t Shamir shares of the truth table corresponding to the point function. More specifically, starting with an $\binom{n}{t}$-party additive DPF which distributes a point function $f_{\alpha,\beta}$ using short keys $\{k_T\}_{T \in \binom{[n]}{t}}$, for each subset $T \in \binom{[n]}{t}$, key k_T is assigned to party P_i satisfying $i \notin T$. By evaluating their DPF keys at hand, all parties can obtain replicated secret sharings and hence degree-t Shamir sharings of the truth table of $f_{\alpha,\beta}$ following the share conversion template described above, which results in a construction of a t-private n-party DPF from a $\binom{n}{t}$-party additive DPF. Intuitively, the security stems from the security of $\binom{n}{t}$-party additive DPF and the privacy of replicated secret sharing scheme.

Construction 1: t-private n-party DPF from $\binom{n}{t}$-party additive DPF

Let $\mathsf{DPF} = (\mathsf{DPF.Gen}, \mathsf{DPF.Eval})$ be an $\binom{n}{t}$-party additive DPF for family of point functions $\mathcal{F} = \{f_{\alpha,\beta} : [D] \to \mathbb{F}\}$.

$\mathsf{DPF}_{n,t}.\mathsf{Gen}(1^\lambda)$:

1. $\mathsf{DPF.Gen}(1^\lambda, \alpha, \beta) \to (K_T)_{T \in \binom{[n]}{t}}$.
2. For each $i \in [n]$, assign DPF keys $k_i \leftarrow \{K_T\}_{T \in \binom{[n]}{t}: i \notin T}$.
3. Output $(k_1, \ldots, k_n)$.

$\mathsf{DPF}_{n,t}.\mathsf{Eval}(i, k_i, x)$:

1. Parse k_i as $\{K_T\}_{T \in \binom{[n]}{t}: i \notin T}$.
2. For each $T \in \{S \in \binom{[n]}{t} : i \notin S\}$, compute $Y_T \leftarrow \mathsf{DPF.Eval}(T, K_T, x)$.
3. Compute its degree-t Shamir share as $y_i \leftarrow \sum_{T \in \binom{[n]}{t}: i \notin T} Y_T \cdot f_T(i)$ and output y_i.

Lemma 19 (Imported lemma [41]: multiparty DPF with additive sharing). *Let $m, d > 0$ be any constant integers. Assume MQ and the dual $\mathbb{F}$-LPN assumptions, there exists a secure,*

m-party DPF with additive sharings $\mathsf{DPF} = (\mathsf{DPF.Gen}, \mathsf{DPF.Eval})$ *that supports the family of point functions* $f_{\alpha,\beta} : [\lambda^d] \to \mathbb{F}$ *with the following characteristics:*

- $\mathsf{DPF.Gen}$ *generates keys of size* $\ell = O(\lambda^3)$.
- $\mathsf{DPF.Gen}$ *can be computed using* $O(\lambda^2 + \mathsf{poly}(\lambda))$ *Boolean operations (where the exponent in* poly *is independent of* d, m*).*
- *For any input* $x \in [\lambda^d]$, $\mathsf{DPF.Eval}(\cdot, x)$ *can be computed by a degree-*d' *polynomial, where* d' *is a constant.*

Lemma 20 (t-private n-party DPF: constant n and t). *Let* n *and* t *such that* $t < n$ *be two integers. Let* $\mathsf{DPF} = (\mathsf{DPF.Gen}, \mathsf{DPF.Eval})$ *be an* $\binom{n}{t}$*-party additive DPF with pseudorandom outputs as per Definition 16 supporting the family of point functions* $f_{\alpha,\beta} : [\lambda^d] \to \mathbb{F}$ *such that*

- $\mathsf{DPF.Gen}$ *generates keys of size* ℓ,
- *for any* $x \in [\lambda^d]$, $\mathsf{DPF.Eval}(\cdot, x)$ *is computable by degree-*d' *polynomials.*

Then the construction in t-private n-party DPF is a secure t*-private* n*-party DPF with degree-*t *Shamir secret sharing with pseudorandom outputs that supports the same family of the point functions with the following characteristics:*

- $\mathsf{DPF}_{n,t}.\mathsf{Gen}$ *generates keys of size* $\binom{n-1}{t} \cdot \ell$.
- $\mathsf{DPF}_{n,t}.\mathsf{Gen}$ *has the same asymptotic computation complexity as that of* $\mathsf{DPF.Gen}$.
- *For any input* $x \in [\lambda^d]$, $\mathsf{DPF}_{n,t}.\mathsf{Eval}(\cdot, x)$ *can be computed by a degree-*d' *polynomial.*

Proof of Lemma 20. The correctness stems from the correctness of $\binom{n}{t}$-party additive scheme DPF $\mathsf{DPF} = (\mathsf{DPF.Gen}, \mathsf{DPF.Eval})$. By invoking $(k_1, \ldots, k_n) \leftarrow \mathsf{DPF}_{n,t}.\mathsf{Gen}(1^\lambda, \alpha, \beta)$ and $y_i \leftarrow \mathsf{DPF}_{n,t}.\mathsf{Eval}(i, k_i, x)$ for each $i \in [n]$, notice that $(y_1, \ldots, y_n)$ forms a degree-t Shamir secret sharing with secret $\sum_{T \in \binom{[n]}{t}} Y_T \cdot f_T(0) = \sum_{T \in \binom{[n]}{t}} Y_T = \sum_{T \in \binom{[n]}{t}} \mathsf{DPF.Eval}(K_T, x) = f_{\alpha,\beta}(x)$.

The security also inherits from that of $\binom{n}{t}$-party additive DPF. For any subset $T \in \binom{[n]}{\leq t}$ of corrupted parties, there exists $R \in \binom{[n]}{t}$ such that $T \subseteq R$, which implies the key K_R labeled by R is not included in the set of $\binom{n}{t}$-DPF keys $\{k_i\}_{i \in T}$ that are known to the corrupted parties. Hence, by security of $\binom{n}{t}$-party additive DPF, there exists a PPT simulator Sim' such that for any point function $f_{\alpha,\beta}$ over domain $[D]$ and finite field $\mathbb{F}$, the distributions

$$\{(K_S)_{S \in \binom{[n]}{t} : S \neq R} \mid (K_S)_{S \in \binom{[n]}{t}} \leftarrow \mathsf{DPF.Gen}(1^\lambda, \alpha, \beta)\}$$

and

$$\{(K_S)_{S \in \binom{[n]}{t} : S \neq R} \mid (K_S)_{S \in \binom{[n]}{t} : S \neq R} \leftarrow \mathsf{Sim}(1^\lambda, D, \mathbb{F})\}$$

are computationally indistinguishable. Then a PPT simulator Sim can be constructed based on Sim' by first invoking Sim' to generate the keys $\{K'_S\}_{S \in \binom{[n]}{t} : S \neq R}$ and then assigning the additive DPF keys to each party P_σ in T by setting $k'_\sigma \leftarrow \{K'_S\}_{S \in \binom{[n]}{t} : \sigma \notin S}$, which is feasible since the key K_R of the $\binom{[n]}{t}$-party additive DPF is absent from the corrupted parties' view. The effectiveness of Sim holds as the distributions

$$\{(k_\sigma)_{\sigma \in T} \mid (k_1, \ldots, k_n) \leftarrow \mathsf{DPF}_{n,t}.\mathsf{Gen}(1^\lambda, \alpha, \beta)\}$$

and

$$\{(k'_\sigma)_{\sigma \in T} \mid (k'_\sigma)_{\sigma \in T} \leftarrow \mathsf{Sim}'(1^\lambda, D, \mathbb{F})\}$$

are computationally indistinguishable.

As for pseudorandomness of the outputs, for any subset $T \in \binom{[n]}{\leq t}$ of parties, by definition, our goal is to prove the following two distributions

$$\left\{ (k_\sigma)_{\sigma \in T}, (y_{\tilde{\sigma}})_{\tilde{\sigma} \in [n] \setminus T} \;\middle|\; \begin{array}{c} (k_1, \ldots, k_n) \leftarrow \mathsf{DPF}_{n,t}.\mathsf{Gen}(1^\lambda, \alpha, \beta) \\ y_{\tilde{\sigma}} \leftarrow \mathsf{DPF}_{n,t}.\mathsf{Eval}(\tilde{\sigma}, k_{\tilde{\sigma}}, x) \text{ for } \tilde{\sigma} \in [n] \setminus T \end{array} \right\}$$

and

$$\left\{ (k_\sigma)_{\sigma\in T}, (y_{\tilde{\sigma}})_{\tilde{\sigma}\in[n]\setminus T} \;\middle|\; \begin{array}{c} (k_1,\ldots,k_n) \leftarrow \mathsf{DPF}_{n,t}.\mathsf{Gen}(1^\lambda,\alpha,\beta) \\ y_\sigma \leftarrow \mathsf{DPF}_{n,t}.\mathsf{Eval}(\sigma,k_\sigma,x) \text{ for } \sigma\in T \\ (y_{\tilde{\sigma}})_{\tilde{\sigma}\in[n]\setminus T} \leftarrow \mathsf{RandShamir}(f_{\alpha,\beta}(x),t,T,\{y_\sigma\}_{\sigma\in T}) \end{array} \right\}$$

are computationally indistinguishable. The statement already holds when $|T| = t$ by correctness. We then consider the case that $|T| < t$ and choose $m := t - |T|$ parties, $P_{i_1},\ldots,P_{i_m}$, that are not in T with m smallest indices. It suffices to prove that for each $j \in [m]$, given the set of keys $\{k_\sigma\}_{\sigma\in T}$ and $\{k_{i_1},\ldots,k_{i_{j-1}}\}$, the output obtained by evaluating $\mathsf{DPF}_{n,t}.\mathsf{Eval}(i_j,k_{i_j},x)$ is pseudorandom. Observe that we only need to show for any $T' \in \binom{[n]}{\leq t-1}$ and any party P_ℓ such that $P_\ell \notin T'$, it holds that given the set of keys $\{k_\sigma\}_{\sigma\in T'}$, the output obtained by evaluating $\mathsf{DPF}_{n,t}.\mathsf{Eval}(\ell,k_\ell,x)$ is pseudorandom (denoted by Statement $*$) since the previous statement can be implied by this by induction on j.

To prove Statement $*$, notice that there exists a subset $R \in \binom{[n]}{t}$ such that $T' \subset R$ but $P_\ell \notin R$, which implies that the additive $\binom{n}{t}$-party DPF K_R labeled by R is known by party P_ℓ but not to parties in R and thus T'. By the pseudorandom output property of additive $\binom{n}{t}$-party DPF, the value $Y_R \leftarrow \mathsf{DPF}.\mathsf{Eval}(R,K_R,x)$ is pseudorandom given the set of keys $\{k_i\}_{i\in T'}$. Therefore, the value

$$y_\ell = \sum_{A\in\binom{[n]}{t}:\ell\notin A} Y_A\cdot f_A(\ell) = Y_R\cdot f_R(\ell) + \sum_{A\in\binom{[n]}{t}:\ell\notin A, A\neq R} Y_A\cdot f_A(\ell)$$

is pseudorandom given the set of keys $\{k_i\}_{i\in T'}$, which proves Statement $*$.

Eventually, we conclude the construction in t-private n-party DPF is a secure t-private n-party DPF with degree-t Shamir secret sharing satisfying the property of pseudorandom output. ■

Observed in [19], there is a general transformation from DPF to PCG for constant-degree correlations assuming hardness of dual-LPN, and further to HSS for constant-degree polynomials due to the equivalence between PCG and HSS. Although such a general transformation is originally considered in the setting of two parties and additive sharings, it can be extended to the setting of multiparty and linear secret sharing schemes. Applying this transformation to a t-private n-party DPF with Σ-sharing results in a t-private n-party PCG for constant-degree correlations with Σ-sharing and a t-private n-party HSS with Σ-sharing for constant-degree polynomials.

Lemma 21 (t-private n-party DPF to PCG). *Let n,t be two integers and d be a constant integer. Let Σ be a t-private linear threshold secret sharing scheme. Assuming the dual $\mathbb{F}$-$\mathsf{LPN}(I,I',k)$ assumption, if there exists a t-private n-party DPF for point functions with domain size $(I')^d$ with Σ-sharing with pseudorandom outputs with key of size ℓ, there exists a secure t-private n-party PCG that generates degree-d correlation $\mathcal{C}(n,d,\Sigma)$, where $\mathcal{C}(n,d,\Sigma)$ is defined as the correlation that all parties take Σ-sharings of each of the components in $\otimes_{j=1}^d \boldsymbol{r}_j$ with each $\boldsymbol{r}_j \in \mathbb{F}^I$ being pseudorandom.*

The seed size of the t-private n-party PCG is $O(k^d\cdot\ell)$, where k denotes the noise weight of the dual $\mathbb{F}$-LPN assumption.

As a side-note, the reverse sampling algorithm for $\mathcal{C}(n,d,\Sigma)$ can be constructed by first sampling the secrets $\{\boldsymbol{r}_j\}_{j\in[d]}$ and invoking the reverse sampling algorithm for Σ-sharing for each component in $\otimes_{j=1}^d \boldsymbol{r}_j$.

Lemma 22 (t-private n-party PCG to HSS). *Let n,t be two integers and d be a constant integer. Let Σ be a t-private linear threshold secret sharing scheme. If there exists a t-private n-party PCG generating degree-d correlation $\mathcal{C}(n,d,\Sigma)$ with seed of size s, there exists a secure t-private n-party HSS with Σ-sharing and pseudorandom outputs that homomorphically evaluates degree-d multi-variate polynomials $P:\mathbb{F}^I\to\mathbb{F}^O$. The share size of the t-private n-party HSS is $O(I+s)$.*

Focusing on existing constructions of multiparty DPF with additive sharings, [24] proposes a construction based on OWFs with key size being exponential in the number of parties. By leveraging randomized bipartite graphs, [55] manages to reduce the exponential dependency of

key size on the number of parties and thus supports a polynomial number of parties. However, both schemes retain a key size that scales with the square root of the domain size. This limitation would hinder their applications to point functions over large domain size.

By leveraging more structured assumptions, [41] builds multiparty DPFs with additive sharings from hardness of LPN and MQ with more compact DPF keys. However, this comes at the cost of supporting only a constant number of parties. Consequently, within this LPN- and MQ-based route, the derived t-private DPF, PCG, and HSS constructions above are presently limited to a constant number of parties. We next explore how to accommodate a polynomial number of parties while preserving compact seeds in the two parameter regimes considered in this work.

5.4 t-private n-party PCG via replicated secret sharings: polynomial n and constant $n-t$

We elaborate our construction of t-private n-party PCGs for constant-degree correlations accommodating polynomial number of parties by first focusing on the case of degree-2 correlations, e.g., Beaver triples, for simplicity. An observation is that letting all parties share the truth table of point functions and thus are able to distributively compute constant-degree correlations (through DPF-to-PCG compiler) is a bit redundant. In particular, suppose all parties first maintain replicated secret sharings of random values a and b, where $a = \sum_{T\in\binom{[n]}{t}} a_T, b = \sum_{S\in\binom{[n]}{t}} b_S$ and each party P_i owns $\{a_T | T \in \binom{[n]}{t}, i \notin T\}, \{b_S | S \in \binom{[n]}{t}, i \notin S\}$. To distribute $a \cdot b = \sum_{T\in\binom{[n]}{t},S\in\binom{[n]}{t}} a_T \cdot b_S$, it remains to distribute each $a_T \cdot b_S$ among parties owning at least one of a_T and b_S, i.e., parties in $M := ([n] \setminus T) \cup ([n] \setminus S)$. Importantly, to maintain the privacy of the resulting t-private n-party PCG, it requires that any subset of parties $\mathcal{C} \subseteq ([n] \setminus T) \cup ([n] \setminus S)$ can reconstruct to learn $a_T \cdot b_S$ only if they are able to reconstruct each of a_T and b_S so that the privacy of b_S (a_T) is preserved when parties only have access to a_T (b_S).

This introduces our need of $(m-(n-t))$-private m-party PCG for constant-degree correlations to distribute $a_T \cdot b_S$ among parties in M, where $m = |M|$ and hence $n - t + 1 \leq m \leq 2 \cdot (n - t)$. Concretely, invoking the $(m-(n-t))$-private m-party PCG to distribute $a_T \cdot b_S$ ensures that if parties in $\mathcal{C}$ do not have access to one of a_T and b_S, say a_T, then $\mathcal{C} \cap ([n] \setminus T) = \emptyset$ and $|\mathcal{C}| \leq m-(n-t)$ (otherwise at least one of parties in $\mathcal{C}$ knows a_T). Hence, by security of $(m-(n-t))$-private m-party PCG, parties in $\mathcal{C}$ have no information about $a_T \cdot b_S$. It is worth noting that programmability of such an $(m-(n-t))$-private m-party PCG that enables programming the pseudorandom variables to be distributed is needed to maintain the consistency among different cross-terms. When $n-t$ is a constant, this can be instantiated by the construction above from multiparty DPF with additive sharing.

Hence, by invoking expanding procedure of $(m-(n-t))$-private m-party PCG, parties in M are able to compute their shares of $[a_T \cdot b_S]^m_{m-(n-t)}$. Observe that if parties outside M take 0 as their shares of $a_T \cdot b_S$, all n shares will define a polynomial with secret equal to $a_T \cdot b_S$ of degree $m-(n-t)+(n-m) = t$. As a result, it can be viewed that all parties hold degree-t Shamir sharing of $a_T \cdot b_S$, i.e., $[a_T \cdot b_S]^n_t$. By linear homomorphism of Shamir secret sharings, all parties can compute their shares of $[a \cdot b]^n_t$.

Based on threshold PCGs accommodating constant number of parties constructed above, when $n-t$ is a constant, we are able to construct a t-private n-party PCG for degree-2 correlations accommodating polynomial number of parties, which can be extended for constant-degree correlations. Both correctness and security follow from that of the replicated secret sharing scheme and the underlying threshold PCG.

Construction 2: t-private n-party PCG for degree-d correlations

Let $\boldsymbol{C}$ be a code generation algorithm such that $H_{I',I} \leftarrow_\$ \boldsymbol{C}(I', I, \mathbb{F})$ is a random parity-check matrix.
$\mathsf{PCG}_{n,t}.\mathsf{Gen}(1^\lambda)$:

1. **Generating d random variables with replicated secret sharings.** For each $j \in [d]$ and $T \in \binom{[n]}{t}$, randomly generate a sparse vector $\boldsymbol{e}_{j,T} \leftarrow_{\$} \mathsf{HW}_w(\mathbb{F}^{I'})$ expanded by $\alpha_{j,T}$. Let $\mathbf{r}_{j,T} \leftarrow H \cdot \mathbf{e}_{j,T}$ and $\mathbf{r}_j \leftarrow \sum_{T \in \binom{[n]}{t}} \mathbf{r}_{j,T}$.
2. **Distributing cross-terms.** For each $(T_1, \ldots, T_d) \in \binom{[n]}{t}^d$, compute
$$(S^i_{T_1,\ldots,T_d})_{i \in M} \leftarrow \mathsf{PCG}_{m,m-(n-t)}.\mathsf{Gen}(1^\lambda, \otimes_{j=1}^d \mathbf{r}_{j,T_j}),$$
where $M := \cup_{j=1}^d ([n] \setminus T_j)$ and $m := |M|$.
3. **Assign seeds for each party.** For each $i \in [n]$, set
$$s_i \leftarrow (\{\alpha_{j,T}\}_{j \in [d], T \in \binom{[n]}{t} : i \notin T}, \{S^i_{T_1,\ldots,T_d}\}_{(T_1,\ldots,T_d) \in \binom{[n]}{t}^d : i \in \cup_{j=1}^d ([n] \setminus T_j)}).$$
4. Output $(s_i)_{i \in [n]}$.

$\mathsf{PCG}_{n,t}.\mathsf{Expand}(i, s_i)$:

1. Parse s_i as
$$s_i \leftarrow (\{\alpha_{j,T}\}_{j \in [d], T \in \binom{[n]}{t} : i \notin T}, \{S^i_{T_1,\ldots,T_d}\}_{(T_1,\ldots,T_d) \in \binom{[n]}{t}^d : i \in \cup_{j=1}^d ([n] \setminus T_j)}).$$
2. For each $j \in [d]$, compute degree-t Shamir sharings of $\mathbf{r}_j$ as
$$[\mathbf{e}_j]^n_t \leftarrow \sum_{T \in \binom{[n]}{t} : i \notin T} \mathbf{e}_{j,T} \cdot f_T(i) \text{ and } [\mathbf{r}_j]^n_t \leftarrow H \cdot [\mathbf{e}_j]^n_t.$$
3. For each $(T_1, \ldots, T_d) \in \binom{[n]}{t}^d$, compute
$$[\otimes_{j=1}^d \mathbf{r}_{j,T_j}]^m_{m-(n-t)} \leftarrow \mathsf{PCG}_{m,m-(n-t)}.\mathsf{Expand}(i, S^i_{T_1,\ldots,T_d}),$$
where $m := |\cup_{j=1}^d ([n] \setminus T_j)|$ and $[\otimes_{j=1}^d \mathbf{r}_{j,T_j}]^m_{m-(n-t)}$ can be viewed as $[\otimes_{j=1}^d \mathbf{r}_{j,T_j}]^n_t$.
4. Compute $[\otimes_{j=1}^d \mathbf{r}_j]^n_t \leftarrow \sum_{(T_1,\ldots,T_d) \in \binom{[n]}{t}^d} [\otimes_{j=1}^d \mathbf{r}_{j,T_j}]^n_t$.
5. Output $(\{[\mathbf{r}_j]^n_t\}_{j \in [d]}, [\otimes_{j=1}^d \mathbf{r}_j]^n_t)$.

Theorem 23 (t-private n-party PCG: polynomial n and constant $n-t$). *Let n, t be integers such that $n-t$ is a constant. Assuming the dual $\mathbb{F}$-LPN assumption, based on existence of $(n'-(n-t))$-private n'-party PCGs with degree-t Shamir secret sharing for all $n - t + 1 \le n' \le d \cdot (n-t)$, the construction in t-private n-party PCG is a secure t-private n-party PCG that generates degree-d correlation $(\{[\overrightarrow{\boldsymbol{r}}_j]^n_t\}_{j \in [d]}, [\otimes^d_{j=1} \overrightarrow{\mathbf{r}_j}]^n_t)$ with degree-t Shamir sharing for n parties, where $\boldsymbol{r}_j \in \mathbb{F}^I$ is pseudorandom for all $j \in [d]$.*

If the construction for multiparty DPF with additive sharings from LPN and MQ in [41] is used to instantiate an $(n' - (n-t))$-private n'-party PCG via Lemma 21, the seed size of the t-private n-party PCG is $O(\binom{n}{t}^d \cdot k^d \cdot \lambda^3)$.

Proof of Theorem 23. We first prove the correctness from the dual LPN assumption and PCG correctness of the underlying PCGs. Starting from the real world distribution

$$\mathcal{D}_0 := \left\{ (R_1, \ldots, R_n) \;\middle|\; \begin{array}{c} (s_1, \ldots, s_n) \leftarrow \mathsf{PCG}_{n,t}.\mathsf{Gen}(1^\lambda) \\ (R_i) \leftarrow \mathsf{PCG}_{n,t}.\mathsf{Expand}(i, s_i) \text{ for each } i \in [n] \end{array} \right\}$$

$$= \left\{ (R_1, \ldots, R_n) \;\middle|\; \begin{array}{c} \boldsymbol{e}_{j,T} \leftarrow \mathsf{HW}_w(\mathbb{F}^{I'}), \boldsymbol{r}_{j,T} \leftarrow H \cdot \boldsymbol{e}_{j,T} \text{ for each } j \in [d], T \in \binom{[n]}{t} \\ \boldsymbol{r}_j \leftarrow \sum_{T \in \binom{[n]}{t}} \boldsymbol{r}_{j,T} \text{ for each } j \in [d] \\ (S^i_{T_1,\ldots,T_d})_{i \in \cup_{j=1}^d ([n] \setminus T_j)} \leftarrow \mathsf{PCG.Gen}(1^\lambda, \otimes_{j=1}^d \boldsymbol{r}_{j,T_j}) \\ s_i \leftarrow (\{\alpha_{j,T}\}_{j \in [d], T \in \binom{[n]}{t} : i \notin T}, \{S^i_{T_1,\ldots,T_d}\}_{(T_1,\ldots,T_d) \in \binom{[n]}{t}^d : i \in \cup_{j=1}^d ([n] \setminus T_j)}) \\ (R_i) \leftarrow \mathsf{PCG}_{n,t}.\mathsf{Expand}(i, s_i) \text{ for each } i \in [n] \end{array} \right\},$$

our goal is to prove $\mathcal{D}_0$ is computationally indistinguishable from the distribution

$$\left\{ (R_1, \ldots, R_n) \;\middle|\; \begin{array}{c} \boldsymbol{r}_j \leftarrow_\$ \mathbb{F}^I \text{ for each } j \in [d] \\ (R_1, \ldots, R_n) \leftarrow \mathsf{RandShamir}(\{\boldsymbol{r}_j\}_{j \in [d]}, \otimes_{j=1}^d \boldsymbol{r}_j, t, \emptyset, \emptyset) \end{array} \right\},$$

which will be shown using the following hybrid arguments.

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

$\mathbf{Hyb}_1$. In this hybrid, for each $(T_1, \ldots, T_d) \in \binom{[n]}{t}^d$, instead of invoking $\mathsf{PCG}_{m,m-(n-t)}.\mathsf{Gen}(1^\lambda, \otimes_{j=1}^d \boldsymbol{r}_{j,T_j})$ to generate seeds, we directly sample random degree-t Shamir sharings with v, for v being each component of $\otimes_{j=1}^d \boldsymbol{r}_{j,T_j}$. The distribution in $\mathbf{Hyb}_1$ is computationally indistinguishable from that in $\mathbf{Hyb}_0$ due to the PCG correctness of $\mathsf{PCG}_{m,m-(n-t)}.\mathsf{Gen}$.

$\mathbf{Hyb}_2$. In this hybrid, for each $j \in [d], T \in \binom{[n]}{t}$, instead of randomly sampling a sparse vector $\boldsymbol{e}_{j,T} \leftarrow_\$ \mathsf{HW}_w(\mathbb{F}^{I'})$ and computing $\boldsymbol{r}_{j,T} \leftarrow H \cdot \boldsymbol{e}_{j,T}$, we directly sample $\boldsymbol{r}_{j,T} \leftarrow_\$ \mathbb{F}^I$. The distribution in $\mathbf{Hyb}_2$ is computationally indistinguishable from that in $\mathbf{Hyb}_1$ assuming the dual-LPN assumption.

$\mathbf{Hyb}_3$. In this hybrid, for each $j \in [d]$, instead of sampling $\boldsymbol{r}_{j,T} \leftarrow_\$ \mathbb{F}^I$ for each $T \in \binom{[n]}{t}$ and computing $\boldsymbol{r}_j \leftarrow \sum_{T \in \binom{[n]}{t}} \boldsymbol{r}_{j,T}$, we directly randomly sample $\boldsymbol{r}_j \leftarrow_\$ \mathbb{F}^I$, $\{\boldsymbol{r}_{j,S}\}_{j \in [d], S \in \binom{[n]}{t}, S \neq T_0}$ and set $\boldsymbol{r}_{j,T_0} = \boldsymbol{r}_j - \sum_{S \in \binom{[n]}{t} : S \neq T_0} \boldsymbol{r}_{j,S}$ for each $j \in [d]$, where $T_0 = \{1, \ldots, t\}$. The distribution in $\mathbf{Hyb}_3$ is identical to that in $\mathbf{Hyb}_2$ due to the local conversion from replicated secret sharing with secret to degree-t Shamir secret sharing and the randomness of $\{\boldsymbol{r}_{j,T_0}\}_{j \in [d]}$ in $\mathbf{Hyb}_2$, which corresponds to the target distribution.

As for the security, we prove this from PCG security of the underlying PCGs. For any set of corrupted parties $T \in \binom{[n]}{\leq t}$, starting from the real world distribution

$$\mathcal{D}_0 := \left\{ (k_\sigma)_{\sigma \in T}, (R_{\tilde{\sigma}})_{\tilde{\sigma} \notin T} \;\middle|\; \begin{array}{c} (S_1, \ldots, S_n) \leftarrow \mathsf{PCG}_{n,t}.\mathsf{Gen}(1^\lambda) \\ R_{\tilde{\sigma}} \leftarrow \mathsf{PCG}_{n,t}.\mathsf{Expand}(\tilde{\sigma}, S_{\tilde{\sigma}}) \text{ for each } \tilde{\sigma} \notin T \end{array} \right\},$$

our goal is to prove this is computationally indistinguishable from the distribution

$$\left\{ (k_\sigma)_{\sigma \in T}, (R_{\tilde{\sigma}})_{\tilde{\sigma} \notin T} \;\middle|\; \begin{array}{c} (S_1, \ldots, S_n) \leftarrow \mathsf{PCG}_{n,t}.\mathsf{Gen}(1^\lambda) \\ R_\sigma \leftarrow \mathsf{PCG}_{n,t}.\mathsf{Expand}(\sigma, S_\sigma) \text{ for each } \sigma \in T \\ (R_{\tilde{\sigma}}) \leftarrow \mathsf{RSample}(T, \{R_\sigma\}_{\sigma \in T}) \end{array} \right\},$$

which will be shown using hybrid arguments.

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

$\mathbf{Hyb}_1$. In this hybrid, for $S \in \binom{[n]}{t}$ such that $T \subseteq S$, for each $\ell \in [d]$ and $(T_1, \ldots, T_d) \in \binom{[n]}{t}$ with $T_\ell = S$, instead of invoking $\mathsf{PCG}_{m,m-(n-t)}.\mathsf{Gen}(1^\lambda, \otimes_{j=1}^d \boldsymbol{r}_{j,T_j})$ to generate PCG seeds for parties in $M := \cup_{j=1}^d ([n] \setminus T_j)$, where $m := |M|$, we invoke the reverse sampling algorithm of $\mathsf{PCG}_{m,m-(n-t)}$. In particular, let $(s^i_{T_1,\ldots,T_d})_{i \in M} \leftarrow \mathsf{PCG}_{m,m-(n-t)}.\mathsf{Gen}(1^\lambda, \otimes_{j=1}^d \boldsymbol{r}_{j,T_j})$, $R^\sigma_{T_1,\ldots,T_d} \leftarrow \mathsf{PCG}_{m,m-(n-t)}.\mathsf{Expand}(\sigma, s^\sigma_{T_1,\ldots,T_d})$ for each $\sigma \in T \cap M$ and $(R^{\tilde{\sigma}}_{T_1,\ldots,T_d})_{\tilde{\sigma} \in M-T} \leftarrow \mathsf{RSample}(T, \{s^\sigma_{T_1,\ldots,T_d}\}_{\sigma \in T \cap M})$. The distribution in $\mathbf{Hyb}_1$ is computationally indistinguishable from that in $\mathbf{Hyb}_0$ due to the PCG security of $\mathsf{PCG}_{m,m-(n-t)}$.

$\mathbf{Hyb}_2$. In this hybrid, instead of invoking the reverse sampling algorithm $\mathsf{RSample}$ of $\mathsf{PCG}_{m,m-(n-t)}$ for each $S \in \binom{[n]}{t}$ such that $T \subseteq S$ and each $(T_1, \ldots, T_d)$ such that there exists some $j \in [d]$ satisfying $T_j = S$ as in $\mathbf{Hyb}_1$, for each component v in $\otimes_{j=1}^d \boldsymbol{r}_j$, we first compute the shares of corrupted parties of $[v]^n_t$ as $\{v_\sigma\}_{\sigma \in T}$ and invoke the reverse sampling algorithm of Shamir secret

sharings $\mathsf{RandShamir}(v, t, T, \{v_\sigma\}_{\sigma\in T})$ to obtain the whole sharing $(v_{\tilde{\sigma}})_{\tilde{\sigma}\in[n]\setminus T}$. The distribution in $\mathbf{Hyb}_2$ is identical to that in $\mathbf{Hyb}_1$.

$\mathbf{Hyb}_3$. In this hybrid, for each $j \in [d]$ and $T \in \binom{[n]}{t}$, instead of randomly sampling a sparse vector $\boldsymbol{e}_{j,T} \leftarrow_\$ \mathsf{HW}_w(\mathbb{F}^{I'})$ and computing $\boldsymbol{r}_{j,T} \leftarrow H \cdot \boldsymbol{e}_{j,T}$, we directly sample $\boldsymbol{r}_{j,T} \leftarrow_\$ \mathbb{F}^I$. The distribution in $\mathbf{Hyb}_3$ is computationally indistinguishable from that in $\mathbf{Hyb}_2$ assuming the dual-LPN assumption.

$\mathbf{Hyb}_4$. In this hybrid, for each $j \in [d]$, instead of sampling $\boldsymbol{r}_{j,T} \leftarrow_\$ \mathbb{F}^I$ for each $T \in \binom{[n]}{t}$ and computing $\boldsymbol{r}_j \leftarrow \sum_{T\in\binom{[n]}{t}} \boldsymbol{r}_{j,T}$, we directly randomly sample $\boldsymbol{r}_j \leftarrow_\$ \mathbb{F}^I$. The distribution in $\mathbf{Hyb}_4$ is identical to that in $\mathbf{Hyb}_3$ due to the local conversion from replicated secret sharing with secret to degree-t Shamir secret sharing, which corresponds to the target distribution. ■

5.5 t-private n-party PCG via party emulation: polynomial n and constant t

To formalize our construction, we consider the following so-called composed secret sharing scheme. To share a secret $s \in \mathbb{F}$, the sharing map $\mathtt{share} : \mathbb{F} \times \mathbb{F}^{1+\binom{n}{t}\cdot t} \to (\mathbb{F}^{\binom{n-1}{t-1}})^{\mathcal{I}}$ first samples a random degree-1 Shamir secret sharing $(s_T)_{T\in\binom{[n]}{t}}$ with secret s distributed among all t-subsets $\{T\}_{T\in\binom{[n]}{t}}$ (this requires $|\mathbb{F}| > \binom{n}{t}$ distinct nonzero evaluation points, which is a polynomial bound on the field size since t is constant). Then for each $T \in \binom{[n]}{t}$, it samples a random additive secret sharing of s_T as $(s_T^i)_{i\in T}$ distributed among all parties in T. Then each party P_i obtains the set of additive shares $S_i \leftarrow (s_T^i)_{T\in\binom{[n]}{t}:i\in T} \in \mathbb{F}^{\binom{n-1}{t-1}}$ that are involved with i as its share of s. The reconstruction map takes as input any subset A of $t+1$ shares, since there are $t+1$ t-subsets contained in A, then $t+1$ shares of $(s_T)_{T\in\binom{[n]}{t}}$ can be recovered and thus s can be reconstructed. As any t shares of s reveal no information about the secret s, this is a t-private threshold secret sharing scheme. The whole sharing $(S_1, \ldots, S_n)$ is denoted by $\langle S\rangle_t$. Notice that this composed secret sharing scheme is linearly homomorphic, i.e., $\langle x_1\rangle_t + \langle x_2\rangle_t = \langle x_1 + x_2\rangle_t$.

To show that composed secret sharing correlation is reverse-sampleable, we define the following procedure which samples a random t-private composed secret sharing given a secret $r \in \mathbb{F}$ and a set of shares $\{R_j\}_{j\in T}$ for some $T \in \binom{[n]}{\le t}$.

- $\mathsf{RandComposed}(r, t, T, \{R_j\}_{j\in T})$: The algorithm first forms a subset $S \in \binom{[n]}{t}$ such that $T \subseteq S$ by further including $t - |T|$ parties with smallest indices that are not in T. Then it randomly samples elements as shares of parties in $S \setminus T$. Then the share r^S labeled with S of degree-1 Shamir sharing can be computed and the whole degree-1 Shamir sharing of r denoted by $(r_{S'})_{S'\in\binom{[n]}{t}}$ is determined by the secret r and the share r_S. Then for each $S' \in \binom{[n]}{t}$ with $S' \neq S$, it randomly samples an additive sharing of $r_{S'}$ according to the secret $r_{S'}$ and known shares with indices in $S \cap S'$. At this point, the t-private composed secret sharing of r denoted by $(R_i)_{i\in[n]}$ is fully determined. Finally, it outputs $(R_j)_{j\in[n]\setminus T}$.

As mentioned in Section 2.3, to construct a t-private n-party DPF and compile it to a t-private n-party PCG for constant-degree correlations, the party emulation framework [58] is used to combine a 1-private $\binom{n}{t}$-party DPF and t-party HSS with additive sharings. More specifically, the point function is first distributed to $\binom{n}{t}$ keys using the 1-private $\binom{n}{t}$-party DPF, where each key corresponds to a subset of t parties. Then parties in each subset apply a t-party HSS scheme to homomorphically evaluate the evaluation function of the $\binom{n}{t}$-party DPF. As a result, the point function is shared among all parties with t-private composed secret sharings. The construction is shown below.

Construction 3: t-private n-party DPF

$\mathsf{DPF}_{n,t}.\mathsf{Gen}(1^\lambda, \alpha, \beta)$:

1. Invoke 1-private $\binom{n}{t}$-party DPF to generate DPF keys. $\mathsf{DPF.Gen}(1^\lambda, \alpha, \beta) \to (k_T)_{T\in\binom{[n]}{t}}$. Denote the function family $\{g_{T,x} : k_T \mapsto \mathsf{DPF.Eval}(T, k_T, x) \mid x \in [D], T \in \binom{[n]}{t}\}$ by $\mathcal{G}$.

2. For each $T \in \binom{[n]}{t}$, invoke t-party additive HSS for $\mathcal{G}$ to generate secret sharings $(x_T^{(i)})_{i\in T} \leftarrow \mathsf{HSS.Share}(1^\lambda, k_T)$.
3. For each $i \in [n]$, set $K_i \leftarrow \{x_T^{(i)}\}_{T\in\binom{[n]}{t}:i\in T}$.
4. Output $(K_i)_{i\in[n]}$.

$\mathsf{DPF}_{n,t}.\mathsf{Eval}(i, K_i, x)$:

1. Parse K_i as $\{x_T^{(i)}\}_{T\in\binom{[n]}{t}:i\in T}$.
2. For each $T \in \{S \in \binom{[n]}{t} : i \in S\}$, compute $y_T^{(i)} \leftarrow \mathsf{HSS.Eval}(i, g_{T,x}, x_T^{(i)})$.
3. Output $Y_i \leftarrow (y_T^{(i)})_{T\in\binom{[n]}{t}:i\in T}$.

Lemma 24 (Imported lemma [6]: 1-private n-party DPF). *Let $n \geq 1$ and ℓ be integers, let $\mathbb{F}$ be a finite field, and let $G : \{0,1\}^\lambda \to \{0,1\}^{2(\lambda+\lceil\log n\rceil)}$ be a PRG computable by constant-degree polynomials. Then, there exists a 1-private n-party DPF for $f_{\alpha,\beta} : \{0,1\}^\ell \to \mathbb{F}$ with the following efficiency features:*

- *Per-party key size $\lambda + (2^{\lceil\log n\rceil} - 1)(\lambda + 2\lceil\log n\rceil)\ell + 2^{\lceil\log n\rceil}(n-1)\lceil\log|\mathbb{F}|\rceil$ bits.*
- Gen *invokes G at most $n(\ell + \lceil\frac{n\log|\mathbb{F}|}{2(\lambda+\lceil\log n\rceil)}\rceil)$ times.*
- Eval *invokes G at most $\ell + \lceil\frac{n\log|\mathbb{F}|}{2(\lambda+\lceil\log n\rceil)}\rceil$ times.*

Theorem 25 (t-private n-party DPF: polynomial n and constant t). *Let n, t be integers. If $\mathsf{DPF} = (\mathsf{DPF.Gen}, \mathsf{DPF.Eval})$ is a secure 1-private $\binom{n}{t}$-party DPF scheme with degree-1 Shamir secret sharing with pseudorandom outputs and $\mathsf{HSS} = (\mathsf{HSS.Share}, \mathsf{HSS.Eval})$ is a t-party HSS with additive sharing for function family $\mathcal{G}$ with pseudorandom outputs, where $\mathcal{G} := \{g_{T,x} : k_T \mapsto \mathsf{DPF.Eval}(T, k_T, x) \mid x \in [D], T \in \binom{[n]}{t}\}$, then the construction in t-private n-party DPF is a secure t-private n-party DPF with the composed linear threshold sharing defined above.*

Proof of Theorem 25. The correctness immediately comes from correctness of the 1-private $\binom{n}{t}$-party DPF scheme and the t-party additive HSS scheme. In fact, notice that letting $y_T := \sum_{i\in T} y_T^{(i)} = \sum_{i\in T} \mathsf{HSS.Eval}(i, x_T^{(i)}) = g_{T,x}(k_T) = \mathsf{DPF.Eval}(T, k_T, x)$ by HSS correctness of HSS and $(y_T)_{T\in\binom{[n]}{t}}$ forms a degree-1 Shamir secret sharing of $f_{\alpha,\beta}(x)$ by DPF correctness of DPF. Hence, $(Y_i)_{i\in[n]}$ forms a t-private composed secret sharing of $f_{\alpha,\beta}(x)$.

As for the security, w.o.l.g., for any set $T \in \binom{[n]}{t}$ of corrupted parties, the PPT. simulator Sim performs as follows. It first invokes the simulator for 1-private $\binom{n}{t}$-party DPF $\mathsf{Sim}_{\mathsf{DPF}}$ to simulate the DPF key k_T corresponding to T. Then it honestly runs $\mathsf{HSS.Share}(1^\lambda, k_T)$ to generate HSS shares $(x_T^{(i)})_{i\in T}$ of input k_T for parties in T. Then for each $S \in \binom{[n]}{t}$ with $S \cap T \neq \emptyset$ and $S \neq T$, it invokes the PPT. simulator for t-party additive HSS $\mathsf{Sim}_{\mathsf{HSS}}$ to simulate HSS shares $(x_S^{(i)})_{i\in S\cap T}$. For each $i \in T$, set $Y_i \leftarrow (x_S^i)_{S\in\binom{[n]}{t}:i\in S}$ Then it outputs $(Y_i)_{i\in T}$. The effectiveness of Sim follows from the HSS security of HSS and the DPF security of DPF.

As for the pseudorandomness of the outputs, for any $T \in \binom{[n]}{\leq t}$, our goal is to prove the following two distributions

$$\mathcal{D}^{\mathsf{real}} := \left\{ (K_i)_{i\in T}, (Y_i)_{i\in[n]\setminus T} \;\middle|\; \begin{array}{c} (k_R)_{R\in\binom{[n]}{t}} \leftarrow \mathsf{DPF.Gen}(1^\lambda, \alpha, \beta) \\ (x_R^{(i)})_{i\in R} \leftarrow \mathsf{HSS.Share}(1^\lambda, k_R, g_{R,x}) \text{ for } R \in \binom{[n]}{t} \\ y_R^{(i)} \leftarrow \mathsf{HSS.Eval}(i, x_R^{(i)}) \text{ for } R \in \binom{[n]}{t}, i \in R - T \\ Y_i \leftarrow (y_R^{(i)})_{R\in\binom{[n]}{t}:i\in R} \text{ for } i \in [n] \setminus T \end{array} \right\}$$

and

$$\mathcal{D}^{\mathsf{ideal}} := \left\{ (K_i)_{i\in T}, (Y'_i)_{i\in[n]\setminus T} \;\middle|\; \begin{array}{c} (k_R)_{R\in\binom{[n]}{t}} \leftarrow \mathsf{DPF.Gen}(1^\lambda, \alpha, \beta) \\ (x_R^{(i)})_{i\in R} \leftarrow \mathsf{HSS.Share}(1^\lambda, k_R, g_{R,x}) \text{ for } R \in \binom{[n]}{t} \\ y_R^{(i)} \leftarrow \mathsf{HSS.Eval}(i, x_R^{(i)}) \text{ for } R \in \binom{[n]}{t}, i \in R \cap T \\ Y_i \leftarrow (y_R^{(i)})_{R\in\binom{[n]}{t}:i\in R} \text{ for } i \in T \\ (Y'_i)_{i\in[n]\setminus T} \leftarrow \mathsf{RandComposed}(f_{\alpha,\beta}(x), t, T, \{Y_i\}_{i\in T}) \end{array} \right\}$$

are computationally indistinguishable.

By pseudorandomness of the outputs of the HSS scheme, we have that the distribution $\mathcal{D}^{\mathsf{real}}$ is computationally indistinguishable with the following distribution

$$\mathcal{D}_1 := \left\{ (K_i)_{i\in T}, (Y'_i)_{i\in[n]\setminus T} \;\middle|\; \begin{array}{c} (k_R)_{R\in\binom{[n]}{t}} \leftarrow \mathsf{DPF.Gen}(1^\lambda, \alpha, \beta) \\ y_R \leftarrow \mathsf{DPF.Eval}(R, k_R, x) \text{ for } R \in \binom{[n]}{t} \\ (x_R^{(i)})_{i\in R} \leftarrow \mathsf{HSS.Share}(1^\lambda, k_R, g_{R,x}) \text{ for } R \in \binom{[n]}{t} \\ y_R^{(i)} \leftarrow \mathsf{HSS.Eval}(i, x_R^{(i)}) \text{ for } R \in \binom{[n]}{t}, i \in R \cap T \\ (y_R'^{(i)})_{i\in R-T} \leftarrow \mathsf{RandAdditive}(y_R, R\cap T, \{y_R^{(i)}\}_{i\in R\cap T}) \text{ for } R \in \binom{[n]}{t} \\ Y'_i \leftarrow (y_R'^{(i)})_{R\in\binom{[n]}{t}:i\in R} \text{ for } i \in [n]\setminus T \end{array} \right\},$$

where $\mathsf{RandAdditive}(x, A, \{x_i\}_{i\in A})$ randomly samples an additive sharing $(x'_i)_{i\in A}$ according to the secret x and a set of shares $\{x_i\}_{i\in A}$ and outputs $(x'_i)_{i\in A}$.

Then by pseudorandomness of the outputs of the DPF scheme, we have that the distribution $\mathcal{D}_1$ is computationally indistinguishable from the following distribution

$$\mathcal{D}_2 := \left\{ (K_i)_{i\in T}, (Y'_i)_{i\in[n]\setminus T} \;\middle|\; \begin{array}{c} (k_R)_{R\in\binom{[n]}{t}} \leftarrow \mathsf{DPF.Gen}(1^\lambda, \alpha, \beta), y_S \leftarrow \mathsf{DPF.Eval}(S, k_S, x) \\ (y'_R)_{R\in\binom{[n]}{t}:R\neq S} \leftarrow \mathsf{RandShamir}(f_{\alpha,\beta}(x), t, \{S\}, y_S) \\ (y_R'^{(i)})_{i\in R-T} \leftarrow \mathsf{RandAdditive}(y_R, R\cap T, \{y_R^{(i)}\}_{i\in R\cap T}) \\ Y'_i \leftarrow (y_R'^{(i)})_{R\in\binom{[n]}{t}:i\in R} \text{ for } i \in [n]\setminus T \end{array} \right\},$$

which corresponds to the distribution $\mathcal{D}^{\mathsf{ideal}}$. ■

6 Applications of Pseudorandom Permutations

Permutation correlations represent an important class of structured correlations beyond additive output sharing, with practical and theoretical significance. They arise naturally in settings where uniqueness, anonymity, or ordering matter, such as anonymous communication and leader election. In this section, we describe two applications of our constructions. The first application is for slot assignment in Dining Cryptographer networks (DC-nets). The second application is to single secret leader election. While our current constructions are limited to the three-party setting, future constructions of PCGs and PCFs for pseudorandom permutations capable of handling more parties would automatically extend these applications to a multi-party setting.

6.1 Anonymous broadcast via DC-nets

In a DC-net [34, 56], parties use secret sharing to hide the provenance of a message. DC-nets form the backbone of anonymous broadcast protocols [1, 38, 48, 64]. The main idea is that each party P_i with message $m_i \in \{0,1\}^\ell$ generates a one-hot vector $\mathbf{v}_i$ with the message m_i in the j-th coordinate, where j is chosen randomly by party P_i. The vector $\mathbf{v}_i$ is then additively secret-shared with all other parties. Given all the secret shares, the parties can locally generate a secret share of the combined vector $\mathbf{v}$ by adding all the secret shared vectors together. That is, they compute a share of $\mathbf{v} := \sum_{i=1}^n \mathbf{v}_i$. Finally, by broadcasting their shares of $\mathbf{v}$ (or posting them to a bulletin board), the parties obtain the vector $\mathbf{v}$ in the clear, which reveals the set of messages while hiding the

provenance of each message. In particular, DC-nets provide unlinkability between parties and their message since the position of each party's message in $\mathbf{v}$ was chosen randomly and independently.

A practical challenge with the DC-net architecture is handling *message collisions*—an artifact of each party randomly selecting the "slot" of $\mathbf{v}$ they write their message into. Concretely, the problem is that if two parties P_i and P_j write their messages in the same non-zero coordinate of the vector $\mathbf{v}$, then the corresponding messages are added together resulting in a "clobbered" output. The standard solution to this problem is to either (1) run a special protocol to allow each party to "reserve" a random slot in the output vector [64, 71] or (2) increase the size of the vector to make the probability of a collision sufficiently small [1, 38]. However, these solutions are undesirable given that they both add communication and computation overheads on the parties. As highlighted by Golle and Juels [56]: "There is no good non-interactive means of enabling all players to select distinct message position".

Here, we show that our constructions of PCGs and PCFs for permutations, provide such a *non-interactive* solution to address collisions, as originally imagined by Golle and Juels: Each party obtains a pseudorandom index automatically assigning them to a unique slot. Therefore, after a one-time setup protocol, the parties can use a DC-net without the risk of collisions. Moreover, this approach avoids the need for increasing the size of the message vector or running an allocation protocol at each round.

While this application to DC-nets is immediate for our unbiased PCF construction, we need to be a little more careful with our biased PCG construction from Appendix B. We show that in the context of DC-nets, where parties broadcast over multiple rounds, we can use a randomness extractor to make the permutation unbiased, while only modestly increasing the communication overhead on the parties (in particular, the parties only need to broadcast a few additional bits in each round, independent of the message).

Dealing with bias via the Von Neumann extractor. We first recall that the bias of the permutation comes from the pseudorandom bit μ being biased towards one. The Von Neumann extractor [73] is a simple way of unbiasing any Bernoulli distribution and works as follows: Given a sequence of biased random bits, examine each consecutive pair of bits and discard all pairs where the xor of the two bits is zero. The remaining bits are then guaranteed to be unbiased. Our idea is to use the multi-round DC-net setting to allow parties to unbias the bits "on the fly" at round r for use in round $r+1$. The main idea is that the parties can use the biased bit PCG from Figure 5 to generate a sequence of $2k$ biased bits at round r. Then, at the end of round r, they reveal k sums corresponding to the sum of each pair of consecutive bits (the sum is computed over $\mathbb{F}_3$). Then, at round $r+1$, the parties use the permutation generated using the first bit μ from a pair whose bits sum to one. By the guarantees of the Von Neumann extractor, all the bits in these pairs are distributed as unbiased pseudorandom bits (if the sum is $1 \in \mathbb{F}_3$, then the two bits in the pair are not equal). Moreover, because the parties only reveal the sum of the bits in the pair, no party learns whether the value of the first bit in the pair is zero or one.[9]

Overall, the protocol proceeds as follows:

One-time Setup. In the one-time setup, the i-th party is given a PCG key k_i for Figure 5. The setup additionally distributes a uniformly random starting permutation to all parties. The round number r is initialized to zero. Given the PCG key k_i, the i-th party runs $\mathsf{PRBits.Expand}$ to generate a sequence of $\ell \cdot 2k$ biased pseudorandom bits and obtains the associated $2k$ pseudorandom biased permutation outputs.

Step 1: Broadcast. In the $r+1$-st round, the parties use the r-th permutation as their slot assignment for the DC-net protocol and compute shares of the vector $\mathbf{v}$ containing all their messages. Then, each party parses the r-th block of $2k$ consecutive shares of bits output by $\mathsf{PRBits.Expand}$ as a sequence of k pairs, and broadcasts the k shares obtained by summing the bits in each pair in this sequence. Each party outputs their share of $\mathbf{v}$ along with their shares of the k pair sums.

Step 2: Generating the next permutation. With the k sums revealed, let j denote the index of the first sum that equals 1 in the sequence of k sums. The parties select the $(2j-1)$-st permutation

[9] Indeed, all they learn is that the two bits are distinct. Moreover, all pairs that sum to 0 or 2 are discarded and therefore do not affect the protocol.

output by PCG.Expand for the r-th round, which corresponds to selecting the pseudorandom permutation generated using the $(2j-1)$-st bit output by PRBits.Expand, i.e., the first bit of the j-th pair.

Analysis. For each pair that sums to 1, the parties do not learn which bit in the pair was zero vs. one. Therefore, the sum of the two consecutive shares reveals whether or not the two bits are equal in $\mathbb{F}_3$, and nothing else. Then, by the properties of the Von Neumann extractor, we have the guarantee that all the pairs that sum to 1 are unbiased. By selecting the first such pair, we have the guarantee that both the bits in that pair are unbiased. Therefore, the permutation associated with the first bit of that sequence is also guaranteed to be unbiased.

Determining the value of k. We still need to analyze how large k needs to be in order to guarantee, with high probability, that at least one pair has two differing bits (so that the sum of the bits is one). We compute k using a simple Chernoff bound. The probability that a bit is 1 is $\frac{2}{3}$. Therefore, the probability of getting two differing consecutive bits is $(\frac{1}{3} \cdot \frac{2}{3}) + (\frac{2}{3} \cdot \frac{1}{3}) = \frac{4}{9}$. Let X be the random variable representing the number of pairs of consecutive different bits in k trials. We want to find k such that $\Pr[X > 0] \geq 1 - 2^{-\lambda}$ so as to guarantee a negligible correctness failure in the security parameter. The expected value of X is $4k/9$ and by the Chernoff bound, we have that: $\Pr[X = 0] \leq e^{-2k/9}$. Therefore, by solving for k such that $2^{-\lambda} \leq e^{-2k/9}$, we get that we need $k \geq \frac{9}{2}\lambda \ln(2)$ so that $\Pr[X > 0] \geq 1 - 2^{-\lambda}$. For $\lambda = 40$, this requires $k \geq 125$ to ensure that the parties can extract a single unbiased bit from the k pairs of bits in each round.

6.2 Single Secret Leader Election

Here, we describe an application of our PCF construction to single secret leader election (SSLE) [8, 15, 31, 32, 36]. The problem of SSLE appears in many distributed systems contexts, and in particular, in proof-of-stake based cryptocurrencies. Informally, SSLE requires a set of parties to secretly agree on a single leader, such that only the elected leader knows that they were elected. Additionally, SSLE requires the leader to be able to prove, at some later point in time, that they were elected. The definition of SSLE is complex and so we refer the reader to Boneh et al. [15] for the full formalization. Here, we sketch how a PCF for pseudorandom permutations coupled with a NIZK [12] gives us a simple SSLE protocol.

The basic idea is that we can let the leader be decided by having the lowest permutation value (i.e., zero). Against a single corruption, the security of the PCF ensures that a non-leader does not learn which of the other two parties was elected in the round. Then, the only missing component is the ability for the elected leader to prove to the other parties that they obtained the lowest permutation value (i.e., that the output of PCF.Eval is zero). This can be achieved by having the output of the setup give all the parties commitments to the set of keys $\{\mathsf{k}_0, \mathsf{k}_1, \mathsf{k}_2\}$, which then allows the i-th party to prove via a NIZK that the output of PCF.Eval (using key k_i) results in zero.

Overall, the three-party SSLE protocol proceeds in three steps:

One-time Setup. In the one-time setup, each party $i \in \{0, 1, 2\}$ is given a PCF key k_i and commitments to all parties' PCF keys k_j for $j \in \{0, 1, 2\}$. The round number r is initialized to zero.

Step 1: Election. In the election step (which is repeated over many rounds), each party runs PCF.Eval using the round number r as input. The party with the zero output is determined as the leader at round r.

Step 2: Proof. In the proof step, the elected leader (say, party i) generates a NIZK to prove that, relative to the commitment of k_i, the output of PCF.Eval using key k_i and input r outputs zero.

Step 3: Verification. The parties use the public commitment of k_i to verify the NIZK generated by party i in round r.

Remark 26 (Requirement for a PCF). We crucially rely on the PCF to ensure that the elected leader can produce an efficient NIZK that doesn't grow with the total number of rounds. If a proof is not required, then a PCG would suffice to instantiate SSLE. Moreover, we note that our (biased) PCG construction gives us a perfectly *unbiased* SSLE protocol (even though the permutation itself is biased) since the party with the zero share is pseudorandomly distributed across all three parties.

Acknowledgements

The authors thank the anonymous reviewers and Yuval Ishai for their helpful comments and suggestions, which improved this paper. Geoffroy Couteau acknowledges the support of the French Agence Nationale de la Recherche (ANR), under the France 2030 ANR Project ANR-22-PECY003 SecureCompute. This work is supported by ERC grant OBELiSC (101115790).

References

1. Ittai Abraham, Benny Pinkas, and Avishay Yanai. Blinder - scalable, robust anonymous committed broadcast. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020: 27th Conference on Computer and Communications Security*, pages 1233–1252, Virtual Event, USA, November 9–13, 2020. ACM Press.
2. 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, *Advances in Cryptology – CRYPTO 2022, Part IV*, volume 13510 of *Lecture Notes in Computer Science*, pages 421–452, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Cham, Switzerland.
3. Damiano Abram, Ariel Nof, Claudio Orlandi, Peter Scholl, and Omer Shlomovits. Low-bandwidth threshold ECDSA via pseudorandom correlation generators. In *2022 IEEE Symposium on Security and Privacy*, pages 2554–2572, San Francisco, CA, USA, May 22–26, 2022. IEEE Computer Society Press.
4. Damiano Abram, Peter Scholl, and Sophia Yakoubov. Distributed (correlation) samplers: How to remove a trusted dealer in one round. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022, Part I*, volume 13275 of *Lecture Notes in Computer Science*, pages 790–820, Trondheim, Norway, May 30 – June 3, 2022. Springer, Cham, Switzerland.
5. Michael Adjedj, Constantin Blokh, Geoffroy Couteau, Arik Galansky, Antoine Joux, and Nikolaos Makriyannis. Two-round 2PC ECDSA at the cost of 1 OLE. Cryptology ePrint Archive, Report 2024/1950, 2024.
6. Amit Agarwal, Elette Boyle, Niv Gilboa, Yuval Ishai, Mahimna Kelkar, and Yiping Ma. Compressing unit-vector correlations via sparse pseudorandom generators. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024, Part VIII*, volume 14927 of *Lecture Notes in Computer Science*, pages 346–383, Santa Barbara, CA, USA, August 18–22, 2024. Springer, Cham, Switzerland.
7. Benny Applebaum and Pavel Raykov. Fast pseudorandom functions based on expander graphs. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B: 14th Theory of Cryptography Conference, Part I*, volume 9985 of *Lecture Notes in Computer Science*, pages 27–56, Beijing, China, October 31 – November 3, 2016. Springer, Berlin, Heidelberg, Germany.
8. Michael Backes, Pascal Berrang, Lucjan Hanzlik, and Ivan Pryvalov. A framework for constructing single secret leader election from MPC. Cryptology ePrint Archive, Report 2022/1040, 2022.
9. Abhishek Banerjee, Chris Peikert, and Alon Rosen. Pseudorandom functions and lattices. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 719–737, Cambridge, UK, April 15–19, 2012. Springer, Berlin, Heidelberg, Germany.
10. Carsten Baum, Lennart Braun, Alexander Munch-Hansen, and Peter Scholl. Moz$\mathbb{Z}_{2^k}$arella: Efficient vector-OLE and zero-knowledge proofs over $\mathbb{Z}_{2^k}$. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022, Part IV*, volume 13510 of *Lecture Notes in Computer Science*, pages 329–358, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Cham, Switzerland.
11. Fabrice Benhamouda, Elette Boyle, Niv Gilboa, Shai Halevi, Yuval Ishai, and Ariel Nof. Generalized pseudorandom secret sharing and efficient straggler-resilient secure computation. In Kobbi Nissim and Brent Waters, editors, *TCC 2021: 19th Theory of Cryptography Conference, Part II*, volume 13043 of *Lecture Notes in Computer Science*, pages 129–161, Raleigh, NC, USA, November 8–11, 2021. Springer, Cham, Switzerland.
12. Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *20th Annual ACM Symposium on Theory of Computing*, pages 103–112, Chicago, IL, USA, May 2–4, 1988. ACM Press.
13. Maxime Bombar, Dung Bui, Geoffroy Couteau, Alain Couvreur, Clément Ducros, and Sacha Servan-Schreiber. FOLEAGE: $\mathbb{F}_4$OLE-based multi-party computation for boolean circuits. In *Advances in Cryptology – ASIACRYPT 2024, Part VI*, Lecture Notes in Computer Science, pages 69–101. Springer, Singapore, Singapore, December 7–11, 2024.

14. Maxime Bombar, Geoffroy Couteau, Alain Couvreur, and Clément Ducros. Correlated pseudorandomness from the hardness of quasi-abelian decoding. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023, Part IV*, volume 14084 of *Lecture Notes in Computer Science*, pages 567–601, Santa Barbara, CA, USA, August 20–24, 2023. Springer, Cham, Switzerland.
15. Dan Boneh, Saba Eskandarian, Lucjan Hanzlik, and Nicola Greco. Single secret leader election. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, pages 12–24, 2020.
16. Charles Bouillaguet, Claire Delaplace, Mickaël Hamdad, and Damien Vergnaud. Practical cryptanalysis of pseudorandom correlation generators based on quasi-abelian syndrome decoding. Cryptology ePrint Archive, Paper 2025/892, 2025.
17. Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018: 25th Conference on Computer and Communications Security*, pages 896–912, Toronto, ON, Canada, October 15–19, 2018. ACM Press.
18. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019: 26th Conference on Computer and Communications Security*, pages 291–308, London, UK, November 11–15, 2019. ACM Press.
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, *Advances in Cryptology – CRYPTO 2019, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 489–518, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Cham, Switzerland.
20. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Correlated pseudorandom functions from variable-density LPN. In *61st Annual Symposium on Foundations of Computer Science*, pages 1069–1080, Durham, NC, USA, November 16–19, 2020. IEEE Computer Society Press.
21. 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, *Advances in Cryptology – CRYPTO 2020, Part II*, volume 12171 of *Lecture Notes in Computer Science*, pages 387–416, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Cham, Switzerland.
22. 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: 24th Conference on Computer and Communications Security*, pages 2105–2122, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press.
23. Elette Boyle, Geoffroy Couteau, and Pierre Meyer. Sublinear-communication secure multiparty computation does not require FHE. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part II*, volume 14005 of *Lecture Notes in Computer Science*, pages 159–189, Lyon, France, April 23–27, 2023. Springer, Cham, Switzerland.
24. Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 337–367, Sofia, Bulgaria, April 26–30, 2015. Springer, Berlin, Heidelberg, Germany.
25. Elette Boyle, Niv Gilboa, and Yuval Ishai. Breaking the circuit size barrier for secure computation under DDH. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 509–539, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Berlin, Heidelberg, Germany.
26. 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: 23rd Conference on Computer and Communications Security*, pages 1292–1303, Vienna, Austria, October 24–28, 2016. ACM Press.
27. Elette Boyle, Niv Gilboa, Yuval Ishai, and Victor I. Kolobov. Programmable distributed point functions. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022, Part IV*, volume 13510 of *Lecture Notes in Computer Science*, pages 121–151, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Cham, Switzerland.
28. Elette Boyle, Niv Gilboa, Yuval Ishai, Huijia Lin, and Stefano Tessaro. Foundations of homomorphic secret sharing. In Anna R. Karlin, editor, *ITCS 2018: 9th Innovations in Theoretical Computer Science Conference*, volume 94, pages 21:1–21:21, Cambridge, MA, USA, January 11–14, 2018. LIPIcs.
29. Elette Boyle, Lisa Kohl, and Peter Scholl. Homomorphic secret sharing from lattices without FHE. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part II*, volume 11477 of *Lecture Notes in Computer Science*, pages 3–33, Darmstadt, Germany, May 19–23, 2019. Springer, Cham, Switzerland.

30. Dung Bui and Geoffroy Couteau. Improved private set intersection for sets with small entries. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *PKC 2023: 26th International Conference on Theory and Practice of Public Key Cryptography, Part II*, volume 13941 of *Lecture Notes in Computer Science*, pages 190–220, Atlanta, GA, USA, May 7–10, 2023. Springer, Cham, Switzerland.
31. Dario Catalano, Dario Fiore, and Emanuele Giunta. Adaptively secure single secret leader election from DDH. In Alessia Milani and Philipp Woelfel, editors, *41st ACM Symposium Annual on Principles of Distributed Computing*, pages 430–439, Salerno, Italy, July 25–29, 2022. Association for Computing Machinery.
32. Dario Catalano, Dario Fiore, and Emanuele Giunta. Efficient and universally composable single secret leader election from pairings. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *PKC 2023: 26th International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 13940 of *Lecture Notes in Computer Science*, pages 471–499, Atlanta, GA, USA, May 7–10, 2023. Springer, Cham, Switzerland.
33. Octavian Catrina and Amitabh Saxena. Secure computation with fixed-point numbers. In Radu Sion, editor, *FC 2010: 14th International Conference on Financial Cryptography and Data Security*, volume 6052 of *Lecture Notes in Computer Science*, pages 35–50, Tenerife, Canary Islands, Spain, January 25–28, 2010. Springer, Berlin, Heidelberg, Germany.
34. David Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1(1):65–75, January 1988.
35. 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: 13th International Conference on Security in Communication Networks*, volume 13409 of *Lecture Notes in Computer Science*, pages 540–563, Amalfi, Italy, September 12–14, 2022. Springer, Cham, Switzerland.
36. Miranda Christ, Kevin Choi, Walter McKelvie, Joseph Bonneau, and Tal Malkin. Accountable secret leader election. In *6th Conference on Advances in Financial Technologies (AFT 2024)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2024.
37. Gil Cohen, Ivan Bjerre Damgård, Yuval Ishai, Jonas Kölker, Peter Bro Miltersen, Ran Raz, and Ron D. Rothblum. Efficient multiparty protocols via log-depth threshold formulae - (extended abstract). In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 185–202, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Berlin, Heidelberg, Germany.
38. Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. Riposte: An anonymous messaging system handling millions of users. In *2015 IEEE Symposium on Security and Privacy*, pages 321–338, San Jose, CA, USA, May 17–21, 2015. IEEE Computer Society Press.
39. Geoffroy Couteau, Lalita Devadas, Srinivas Devadas, Alexander Koch, and Sacha Servan-Schreiber. QuietOT: Lightweight oblivious transfer with a public-key setup. In *Advances in Cryptology – ASIACRYPT 2024, Part II*, Lecture Notes in Computer Science, pages 197–231. Springer, Singapore, Singapore, December 7–11, 2024.
40. Geoffroy Couteau and Naman Kumar. 10-party sublinear secure computation from standard assumptions. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024, Part IX*, volume 14928 of *Lecture Notes in Computer Science*, pages 39–73, Santa Barbara, CA, USA, August 18–22, 2024. Springer, Cham, Switzerland.
41. Geoffroy Couteau, Naman Kumar, and Xiaxi Ye. Multiparty Homomorphic Secret Sharing and More from LPN and MQ. *Cryptology ePrint Archive*, 2025.
42. Geoffroy Couteau, Pierre Meyer, Alain Passelègue, and Mahshid Riahinia. Constrained pseudorandom functions from homomorphic secret sharing. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part III*, volume 14006 of *Lecture Notes in Computer Science*, pages 194–224, Lyon, France, April 23–27, 2023. Springer, Cham, Switzerland.
43. Ronald Cramer, Ivan Damgård, and Yuval Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In Joe Kilian, editor, *TCC 2005: 2nd Theory of Cryptography Conference*, volume 3378 of *Lecture Notes in Computer Science*, pages 342–362, Cambridge, MA, USA, February 10–12, 2005. Springer, Berlin, Heidelberg, Germany.
44. Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In Shai Halevi and Tal Rabin, editors, *TCC 2006: 3rd Theory of Cryptography Conference*, volume 3876 of *Lecture Notes in Computer Science*, pages 285–304, New York, NY, USA, March 4–7, 2006. Springer, Berlin, Heidelberg, Germany.
45. Quang Dao, Yuval Ishai, Aayush Jain, and Huijia Lin. Multi-party homomorphic secret sharing and sublinear mpc from sparse lpn. In *Annual International Cryptology Conference*, pages 315–348. Springer, 2023.

46. Samuel Dittmer, Yuval Ishai, and Rafail Ostrovsky. Line-point zero knowledge and its applications. LIPIcs, pages 5:1–5:24. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2021.
47. Yevgeniy Dodis, Shai Halevi, Ron D. Rothblum, and Daniel Wichs. Spooky encryption and its applications. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part III*, volume 9816 of *Lecture Notes in Computer Science*, pages 93–122, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Berlin, Heidelberg, Germany.
48. Saba Eskandarian, Henry Corrigan-Gibbs, Matei Zaharia, and Dan Boneh. Express: Lowering the cost of metadata-hiding communication with cryptographic privacy. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021: 30th USENIX Security Symposium*, pages 1775–1792. USENIX Association, August 11–13, 2021.
49. Sebastian Faust, Carmit Hazay, David Kretzler, Leandro Rometsch, and Benjamin Schlosser. Non-interactive threshold bbs+ from pseudorandom correlations. In *Cryptographers' Track at the RSA Conference*, pages 198–222. Springer, 2025.
50. Nelly Fazio, Rosario Gennaro, Tahereh Jafarikhah, and William E Skeith. Homomorphic secret sharing from paillier encryption. In *Provable Security: 11th International Conference, ProvSec 2017, Xi'an, China, October 23-25, 2017, Proceedings 11*, pages 381–399. Springer, 2017.
51. Ingerid Fosli, Yuval Ishai, Victor I. Kolobov, and Mary Wootters. On the download rate of homomorphic secret sharing. In Mark Braverman, editor, *ITCS 2022: 13th Innovations in Theoretical Computer Science Conference*, volume 215, pages 71:1–71:22, Berkeley, CA, USA, January 31 – February 3, 2022. LIPIcs.
52. Ximing Fu, Mo Li, Shihan Lyu, and Chuanyi Liu. Bit-fixing correlation attacks on Goldreich's pseudorandom generators. Cryptology ePrint Archive, Report 2024/1594, 2024.
53. Niv Gilboa and Yuval Ishai. Compressing cryptographic resources. In Michael J. Wiener, editor, *Advances in Cryptology – CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 591–608, Santa Barbara, CA, USA, August 15–19, 1999. Springer, Berlin, Heidelberg, Germany.
54. Niv Gilboa and Yuval Ishai. Distributed point functions and their applications. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 640–658, Copenhagen, Denmark, May 11–15, 2014. Springer, Berlin, Heidelberg, Germany.
55. Aarushi Goel, Mingyuan Wang, and Zhiheng Wang. Multiparty distributed point functions. In Yael Tauman Kalai and Seny F. Kamara, editors, *Advances in Cryptology – CRYPTO 2025*, pages 140–173, Cham, 2025. Springer Nature Switzerland.
56. Philippe Golle and Ari Juels. Dining cryptographers revisited. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 456–473, Interlaken, Switzerland, May 2–6, 2004. Springer, Berlin, Heidelberg, Germany.
57. Shai Halevi, Yuval Ishai, Abhishek Jain, Eyal Kushilevitz, and Tal Rabin. Secure multiparty computation with general interaction patterns. In Madhu Sudan, editor, *ITCS 2016: 7th Conference on Innovations in Theoretical Computer Science*, pages 157–168, Cambridge, MA, USA, January 14–16, 2016. Association for Computing Machinery.
58. Martin Hirt and Ueli Maurer. Player simulation and general adversary structures in perfect multiparty computation. *J. Cryptol.*, 13(1):31–60, January 2000.
59. Zhicong Huang, Wen jie Lu, Cheng Hong, and Jiansheng Ding. Cheetah: Lean and fast secure two-party deep neural network inference. In Kevin R. B. Butler and Kurt Thomas, editors, *USENIX Security 2022: 31st USENIX Security Symposium*, pages 809–826, Boston, MA, USA, August 10–12, 2022. USENIX Association.
60. Yuval Ishai, Mahimna Kelkar, Varun Narayanan, and Liav Zafar. One-message secure reductions: On the cost of converting correlations. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023, Part I*, volume 14081 of *Lecture Notes in Computer Science*, pages 515–547, Santa Barbara, CA, USA, August 20–24, 2023. Springer, Cham, Switzerland.
61. Yashvanth Kondi, Claudio Orlandi, and Lawrence Roy. Two-round stateless deterministic two-party Schnorr signatures from pseudorandom correlation functions. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023, Part I*, volume 14081 of *Lecture Notes in Computer Science*, pages 646–677, Santa Barbara, CA, USA, August 20–24, 2023. Springer, Cham, Switzerland.
62. Zhe Li, Chaoping Xing, Yizhou Yao, and Chen Yuan. Efficient pseudorandom correlation generators for any finite field. Lecture Notes in Computer Science, pages 145–175. Springer, Cham, Switzerland, June 2025.
63. Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. *Journal of the ACM*, 51(2):231–262, March 2004.
64. Zachary Newman, Sacha Servan-Schreiber, and Srinivas Devadas. Spectrum: High-bandwidth anonymous broadcast. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 229–248, 2022.

65. 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, *Advances in Cryptology – EUROCRYPT 2021, Part I*, volume 12696 of *Lecture Notes in Computer Science*, pages 678–708, Zagreb, Croatia, October 17–21, 2021. Springer, Cham, Switzerland.
66. Srinivasan Raghuraman and Peter Rindal. Blazing fast PSI from improved OKVS and subfield VOLE. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022: 29th Conference on Computer and Communications Security*, pages 2505–2517, Los Angeles, CA, USA, November 7–11, 2022. ACM Press.
67. Mayank Rathee, Conghao Shen, Sameer Wagh, and Raluca Ada Popa. ELSA: Secure aggregation for federated learning with malicious actors. In *2023 IEEE Symposium on Security and Privacy*, pages 1961–1979, San Francisco, CA, USA, May 21–25, 2023. IEEE Computer Society Press.
68. Peter Rindal and Phillipp Schoppmann. VOLE-PSI: Fast OPRF and circuit-PSI from vector-OLE. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, Part II*, volume 12697 of *Lecture Notes in Computer Science*, pages 901–930, Zagreb, Croatia, October 17–21, 2021. Springer, Cham, Switzerland.
69. Lawrence Roy and Jaspal Singh. Large message homomorphic secret sharing from DCR and applications. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part III*, volume 12827 of *Lecture Notes in Computer Science*, pages 687–717, Virtual Event, August 16–20, 2021. Springer, Cham, Switzerland.
70. Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.
71. Chris Studholme and Ian Blake. Multiparty computation to generate secret permutations. Cryptology ePrint Archive, Report 2007/353, 2007.
72. The OpenSSL Project. OpenSSL cryptography and SSL/TLS toolkit. https://www.openssl.org/, 2024. Accessed: 2024-02-12.
73. John Von Neumann et al. Various techniques used in connection with random digits. *John von Neumann, Collected Works*, 5:768–770, 1963.
74. Chenkai Weng, Kang Yang, Jonathan Katz, and Xiao Wang. Wolverine: Fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. In *2021 IEEE Symposium on Security and Privacy*, pages 1074–1091, San Francisco, CA, USA, May 24–27, 2021. IEEE Computer Society Press.
75. Kang Yang, Pratik Sarkar, Chenkai Weng, and Xiao Wang. QuickSilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021: 28th Conference on Computer and Communications Security*, pages 2986–3001, Virtual Event, Republic of Korea, November 15–19, 2021. ACM Press.
76. Kang Yang, Chenkai Weng, Xiao Lan, Jiang Zhang, and Xiao Wang. Ferret: Fast extension for correlated OT with small communication. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020: 27th Conference on Computer and Communications Security*, pages 1607–1626, Virtual Event, USA, November 9–13, 2020. ACM Press.

Supplementary Materials

Table of Contents

On Compressing Linearly Shared Correlations . . . 1
Geoffroy Couteau[1], Alexander Koch[2], Nikolas Melissaris[1], Peter Scholl[3], Sacha Servan-Schreiber[4], and Xiaxi Ye[5]
1 Introduction . . . 2
1.1 Beyond additive output sharing . . . 2
1.2 Our contributions . . . 3
1.3 Applications . . . 5
2 Technical Overview . . . 6
2.1 Background . . . 6
2.2 The degree-1 Shamir-output HSS building block and its applications . . . 7
2.3 PCGs for threshold-shared correlations . . . 9
3 Preliminaries . . . 11
3.1 Homomorphic Secret Sharing . . . 11
3.2 Function Secret Sharing and Distributed Point Functions . . . 13
3.3 Pseudorandom Correlation Generators . . . 14
3.4 Pseudorandom Correlation Functions . . . 15
3.5 Linear-Output Correlations . . . 15
4 PCF for Permutations . . . 16
5 Multiparty PCG for Correlations with Threshold Secret Sharings . . . 20
5.1 Shamir secret sharing and reverse sampling . . . 22
5.2 Multiparty DPF, PCG, HSS with threshold secret sharing schemes . . . 22
5.3 t-private n-party PCG for constant-degree polynomials from additive DPF: constant n and t . . . 24
5.4 t-private n-party PCG via replicated secret sharings: polynomial n and constant $n-t$ 27
5.5 t-private n-party PCG via party emulation: polynomial n and constant t . . . 30
6 Applications of Pseudorandom Permutations . . . 32
6.1 Anonymous broadcast via DC-nets . . . 32
6.2 Single Secret Leader Election . . . 34
A Definition of Pseudorandom Correlation Functions . . . 40
B Constructing a PCG for ε-Biased Permutations . . . 41
B.1 High-Level Overview . . . 42
B.2 Additional Background . . . 42
B.3 Permutation PCG From PCG for Biased Bits . . . 43
B.4 Instantiating the (1/6)-Biased Bits PCG from Quasi-Abelian Syndrome Decoding 45
B.5 Implementation and parameters . . . 49
B.6 Benchmarks . . . 49
B.7 Performance Estimate for the Unbiased Construction . . . 49

A Definition of Pseudorandom Correlation Functions

Here, we include the formal definition of a PCF.

Definition 27 (Pseudorandom Correlation Function [20]). *Let λ be a security parameter, $\mathcal{C}$ be a reverse-sampleable n-correlation with output length $\ell(\lambda) \in \mathsf{poly}(\lambda)$, and $\lambda \leq k = k(\lambda) \in \mathsf{poly}(\lambda)$ be an input length. A Pseudorandom Correlation Function (PCF) for $\mathcal{C}$ is defined by a pair of algorithms* $(\mathsf{PCF.KeyGen}, \mathsf{PCF.Eval})$ *with the following functionality:*

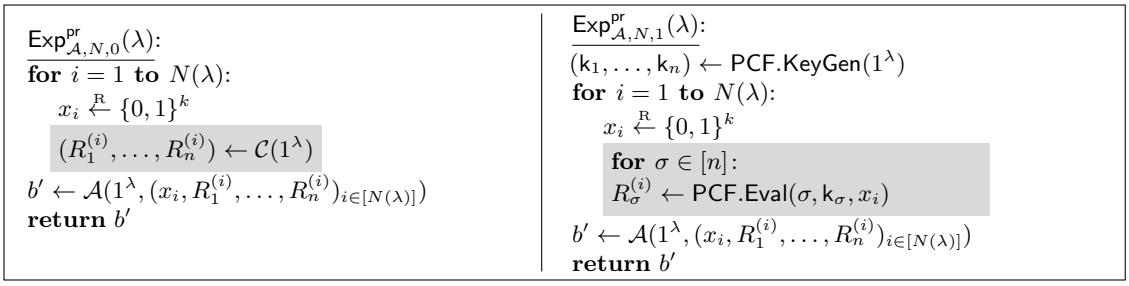


Fig. 3: Pseudorandom $\mathcal{C}$-correlated outputs of a PCF.

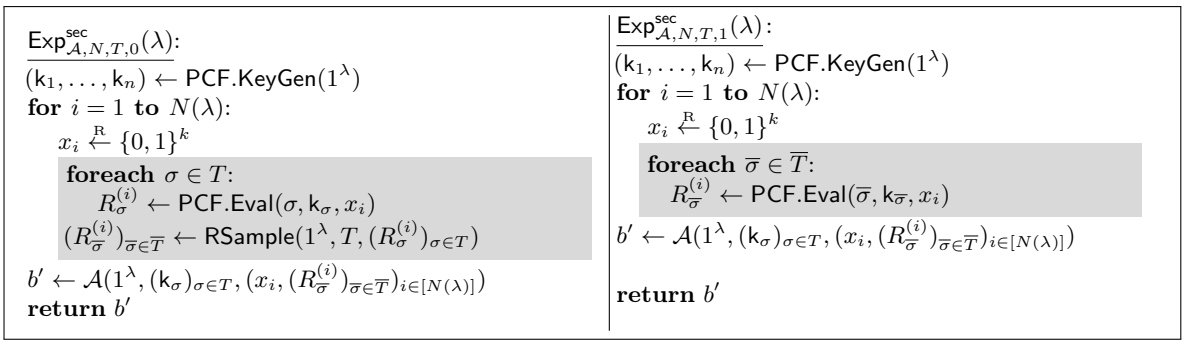


Fig. 4: Security game for a PCF, with a subset $T \subset [n]$ of corrupted parties. Here, $\mathsf{RSample}$ is the algorithm for reverse sampling the correlation $\mathcal{C}$, as defined in Definition 7.

- $\mathsf{PCF.KeyGen}(1^\lambda) \to (\mathsf{k}_1, \ldots, \mathsf{k}_n)$. *Takes as input the security parameter λ. Outputs an n-tuple of keys $(\mathsf{k}_1, \ldots, \mathsf{k}_n)$.*
- $\mathsf{PCF.Eval}(\sigma, \mathsf{k}_\sigma, x) \to R_\sigma$. *Takes as input $\sigma \in [n]$, a key k_σ, and input string $x \in \{0,1\}^k$. Outputs a string $R_\sigma \in \{0,1\}^\ell$.*

We say $\mathsf{PCF} = (\mathsf{PCF.KeyGen}, \mathsf{PCF.Eval})$ *is a (weak) PCF for the correlation $\mathcal{C}$, if the following properties hold:*

Correctness / Pseudorandom $\mathcal{C}$-correlated outputs. *For every $\sigma \in [n]$, all efficient adversaries $\mathcal{A}$, and all $N \in \mathsf{poly}(\lambda)$, there exists a negligible function* negl *such that for all sufficiently large λ,*

$$\left|\Pr[\mathsf{Exp}^{\mathsf{pr}}_{\mathcal{A},N,0}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{pr}}_{\mathcal{A},N,1}(\lambda) = 1]\right| \leq \mathsf{negl}(\lambda),$$

where $\mathsf{Exp}^{\mathsf{pr}}_{\mathcal{A},N,b}(\lambda)$, for $b \in \{0,1\}$, is as defined in Figure 3. In particular, the adversary is given access to $N(\lambda)$ samples.

Security. *For all $T \subset [n]$ and all efficient adversaries $\mathcal{A}$, there exists a negligible function* negl *such that for all sufficiently large λ,*

$$\left|\Pr[\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,T,0}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,T,1}(\lambda) = 1]\right| \leq \mathsf{negl}(\lambda),$$

where $\mathsf{Exp}^{\mathsf{sec}}_{\mathcal{A},N,T,b}(\lambda)$, for $b \in \{0,1\}$, is as defined in Figure 4 (again, with the adversary given $N(\lambda)$ samples).

B Constructing a PCG for ε-Biased Permutations

In this section, we present our 3-party PCG for ε-biased permutations, with $\varepsilon = 1/6$, based on the quasi-abelian syndrome decoding assumption.

B.1 High-Level Overview

Recall the template in Section 2.2: for Alice, Bob and Carol to obtain a permutation, it suffices for Alice and Bob to obtain random, additive shares of a random bit μ, while Alice and Carol obtain shares of the complement, $1-\mu$, in such a way that Alice's share has the *same* value in both instances.

New tool: "Doubly-programmable" PCG for random shared bits. To enable this distribution of shares, we introduce a form of a two-party PCG with a new, programmable property. Previous works [19, 21] used a programmable property that allows to generate new PCG seeds that expand to give the same (or, a related), additively shared *value*, as the original seeds, albeit with fresh shares. We extend this to "one-more-level" of programmability by making the seed (resp. key) of one party sampled independently of the correlation. This makes it possible to fix a seed/key k_0 and then generate multiple keys $\mathsf{k}_1, \mathsf{k}_2$ for two different programmable correlations $\mathcal{C}_1$ and $\mathcal{C}_2$, respectively. More concretely, coming back to our use case of pseudorandom shared bits, Alice can be given k_0 which allows her to generate a pseudorandom share x while Bob and Carol can be given k_1 and k_2, respectively, such that:

$$\mathsf{Expand}(\mathsf{k}_0) - \mathsf{Expand}(\mathsf{k}_1) = \mu \text{ and } \mathsf{Expand}(\mathsf{k}_0) - \mathsf{Expand}(\mathsf{k}_2) = 1 - \mu.$$

Formally, we define such "doubly-programmable" PCGs and PCFs as being $\mathcal{F}$-programmable, where (1) one of the seeds/keys can be sampled independently of the ultimate correlation and (2) the correlation itself can be programmed to be some function of the source of pseudorandomness. (In the above example, the programmed function in k_2 maps x to $1-x$.)

Challenges in generating bits over a field. Generating a pseudorandom bit shared over a larger field is useful for efficient multi-party computation protocols [33, 44, 60] as it enables efficient conversion of secret shares from one field to another. A generic feasibility route for the biased-bit correlation considered below already follows from LPN-based PCGs for constant-degree correlations. In particular, the general construction of Boyle et al. [19] applies over finite fields, albeit with poor concrete computation, and more recent work gives programmable PCGs for OLE over any finite field [62]. Thus, the quasi-abelian syndrome decoding assumption is not needed merely to obtain feasibility of a biased permutation PCG. Our use of QA-SD is instead to instantiate the programmable template below using the small-field PCG techniques of Bombar et al. [14], with the goal of obtaining a more concrete construction; we discuss the current concrete-security caveat for this instantiation in Section B.5.

To avoid relying on high-degree HSS, we observe that in the case of $\mathbb{F}_3$, a shared, random bit can be obtained from just one OLE correlation in $\mathbb{F}_3$, provided we settle for shares of a *biased* bit in $\mathbb{F}_3$. In particular, consider an OLE correlation $((x_0, y_0), (x_1, y_1))$, where $y_1 = y_0 + x_0 x_1 \bmod 3$. The two parties can define shares of x^2, where $x := x_0 + x_1$, by letting $z_0 := x_0^2 - 2y_0$ and $z_1 := x_1^2 + 2y_1$. We then have, $z_0 + z_1 = x_0^2 + x_1^2 + 2x_0x_1 = x^2$. If x is uniform in $\mathbb{F}_3$, then a simple case analysis shows that x^2 is a bit that equals 1 with probability 2/3, and 0 otherwise.

This observation forms the backbone of our PCG construction, which we show can be instantiated by building off of existing PCGs for OLE correlations in $\mathbb{F}_3$.

In this section, we present the framework for building a PCG/PCF for pseudorandom permutations from a PCG/PCF for pseudorandom bits and then instantiate it from the QA-SD assumption and HSS. In Appendix B.2, we first describe the $\mathcal{F}$-programmability definition that we will make use of in our constructions.

B.2 Additional Background

Programmable FSS. We use 2-party FSS schemes with a special *programmability* property, meaning that one of the two keys can be sampled independently of the function f being shared. The following definition extends the concept of programmable distributed point function [27] to programmable FSS for general functions.

Definition 28 (Programmable FSS). *We say that an FSS scheme* $(\mathsf{FSS.Gen}, \mathsf{FSS.Eval})$ *is* programmable *if* $\mathsf{FSS.Gen}$ *can be split into two algorithms* Gen_0 *and* Gen_1 *with the following syntax:*

– $\mathsf{Gen}_0(1^\lambda, I, \mathbb{G}) \to \mathsf{k}_0$. *On input the security parameter and input/output domains of the function* f_λ*, this samples a key* k_0.
– $\mathsf{Gen}_1(\mathsf{k}_0, f_\lambda) \to \mathsf{k}_1$. *on input key* k_0 *and the function* f_λ*, this generates a key* k_1.

It is easy to see that a programmable DPF immediately also implies programmable FSS for sums of t point functions.

Boyle et al. [27] construct a programmable distributed point function for any domain size $N = N(\lambda) \in \mathsf{poly}(\lambda)$ based on a pseudorandom generator. Their construction starts by building a simple and efficient, but *leaky*, programmable DPF, and then upgrades this to a fully secure construction through privacy amplification techniques. While the final construction is not practical, in Appendix B.5 we show how to leverage the practical, leaky version of their DPF in our construction to build a concretely efficient $\mathcal{F}$-programmable PCG.

Doubly-Programmable PCGs. We now introduce our notion of doubly-programmable PCGs. We consider the following class of two-party correlations, where the parties' outputs form subtractive shares of some random variable.

Definition 29 (Two-party subtractive correlation generator). *Let* $(\mathbb{G}, +)$ *be an abelian group and* $\ell, n \in \mathbb{Z}$ *be functions of* λ*. Further, let* $\mathcal{F} = \{f_\lambda \colon \mathbb{G}^\ell \to \mathbb{G}^n\}_\lambda$ *be a family of functions. A* subtractive correlation generator *for* $\mathcal{F}$ *is a correlation generator* $\mathcal{C}_\mathcal{F}$*, such that* $\mathcal{C}_\mathcal{F}(1^\lambda)$ *uniformly samples* $X \leftarrow \mathbb{G}^\ell$ *and outputs* $(R_0, R_1) \in \mathbb{G}^n \times \mathbb{G}^n$*, such that* R_0 *and* R_1 *are distributed uniformly, conditioned on* $R_0 - R_1 = f_\lambda(X)$.

A doubly-programmable PCG allows PCG keys to be generated in a special way, such that both (1) the randomness $f_\lambda(X)$, and (2) the R_0 output of party 0, can be reused across multiple instances. Furthermore, we allow a small "tweak" to be applied to $f_\lambda(X)$ such that it can be modified slightly when reusing it. We formalise this notion by using a special seed from which the randomness in $f_\lambda(X)$ is derived, and an additional input to party 1's key generation algorithm for performing the tweak specified by some function from a family $\mathcal{F}$.

Definition 30 ($\mathcal{F}$-Programmable). *A two-party PCG (resp. PCF) for a subtractive correlation generator (Definition 29) is* $\mathcal{F}$-programmable *for a class of functions* $\mathcal{F}$*, if* $\mathsf{PCG.Gen}$ *(resp.* $\mathsf{PCF.Gen}$*) can be split into two distinct algorithms:*

– $\mathsf{Gen}_0(1^\lambda)$*: a PPT algorithm that on input* 1^λ*, outputs a key* $\mathsf{k}_0 \in \{0,1\}^*$
– $\mathsf{Gen}_1(\mathsf{k}_0, \mathsf{seed}, f)$*: a deterministic algorithm that on input a key* k_0*,* $\mathsf{seed} \in \{0,1\}^\lambda$ *and function* $f \in \mathcal{F}$*, derives and outputs a second key* k_1 *(all randomness being derived from* seed*).*

We require the following correctness property, for any $f \in \mathcal{F}$*:*

$$\Pr\left[R_0 - R_2 = f(R_0 - R_1) : \begin{array}{r} \mathsf{seed} \xleftarrow{\$} \{0,1\}^\lambda \\ \mathsf{k}_0 \xleftarrow{\$} \mathsf{PCG.Gen}_0(1^\lambda) \\ \mathsf{k}_1 \leftarrow \mathsf{PCG.Gen}_1(\mathsf{k}_0, \mathsf{seed}, \mathsf{id}) \\ \mathsf{k}_2 \leftarrow \mathsf{PCG.Gen}_1(\mathsf{k}_0, \mathsf{seed}, f) \\ R_j \leftarrow \mathsf{PCG.Expand}(\mathsf{k}_j),\ j \in \{0,1,2\} \end{array}\right] \geq 1 - \mathsf{negl}(\lambda)$$

The $\mathsf{PCG.Gen}$ (resp. $\mathsf{PCF.Gen}$) algorithm then simply runs Gen_0, followed by Gen_1 on input a random seed and f as the identity function, to output a pair of keys $(\mathsf{k}_0, \mathsf{k}_1)$.

B.3 Permutation PCG From PCG for Biased Bits

We now show how to obtain a PCG for biased, three-party permutation correlations, given a special form of two-party PCG for secret-shared, biased random bits.

Definition 31 (Two-party, ε-biased bits correlation generator). *An* ε-biased bits correlation generator *is a subtractive correlation generator* $\mathcal{C}$ *(from Definition 29), such that* $\mathcal{C}(1^\lambda)$ *samples a vector* $\mathbf{x} \leftarrow \{0,1\}^\ell$ *where each element* x_i *is sampled from* $\mathsf{Bernoulli}(1/2 + \varepsilon)$ *and outputs* $(\mathbf{R}_0, \mathbf{R}_1) \in \mathbb{Z}_q^\ell \times \mathbb{Z}_q^\ell$*, such that the* $\mathbf{R}_i$*'s are distributed uniformly, conditioned on* $\mathbf{R}_0 - \mathbf{R}_1 = \mathbf{x}$.

Definition 32 ($\mathcal{D}$-biased, n-party permutation correlation generator). *A $\mathcal{D}$-biased, n-party permutation correlation generator is a correlation generator that samples ℓ permutations $(\pi_1, \ldots, \pi_\ell) \leftarrow \mathcal{D}^\ell$ where $\mathcal{D}$ is a distribution on the set of all permutations on n elements, S_n. It outputs $\mathbf{R}_1, \ldots, \mathbf{R}_n$, where $\mathbf{R}_i(j) = \pi_j[i]$, meaning that in the j-th position of $\mathbf{R}_i$ there is the i-th element of permutation π_j.*

Lemma 33. *Let $\ell \in \mathsf{poly}(\lambda)$ and let n be constant. Let $\mathcal{C}$ be a $\mathcal{D}$-biased, n-party permutation correlation generator for a fixed, efficiently sampleable distribution $\mathcal{D}$ on S_n (independent of the security parameter). Then, $\mathcal{C}$ is a reverse-sampleable t-private n-party correlation generator for any $t < n$.*

Proof. For this, we need to show that there is an efficient algorithm $\mathsf{RSample}$ that takes as input a party subset $T \subset [n]$ and their correlation samples $R_\sigma =: (e_{\sigma,1}, \ldots, e_{\sigma,\ell})$ with $e_i \in [n]$, and outputs a list of samples $(R_i)_{i \in \overline{T}}$ that fulfills Definition 7. $\mathsf{RSample}$ proceeds by rejection sampling of $\mathcal{D}$ as follows. For each $j \in [\ell]$, it repeatedly samples a permutation π_j using $\mathcal{D}$, until $\pi_j(i) = e_i$ for all $i \in T$. Since n and $\mathcal{D}$ are fixed, every permutation in the support of $\mathcal{D}$ has a fixed positive probability. For any conditioning event that can occur in the real experiment, the acceptance probability is therefore bounded below by a positive constant, so the rejection sampler runs in expected polynomial time. Its output is exactly distributed as $\mathcal{D}$ conditioned on the corrupted parties' coordinates, and hence the resulting correlation is identically distributed to the real one. ■

When we build a $\mathcal{D}$-biased correlation generator based on ε-biased bits, this will give the *following distribution* $\mathcal{D}_\varepsilon$ on S_3, the set of permutations on $\mathbb{F}_3$. It builds a permutation π by sampling a biased bit $\mu \xleftarrow{R} \mathsf{Bernoulli}(1/2 + \varepsilon)$ and a uniformly random $i \xleftarrow{R} \mathbb{F}_3$, and setting $\pi := (i,\ i+1,\ i+2)$ if $\mu = 1$, and $\pi := (i,\ i+2,\ i+1)$ otherwise, where we write $\pi = (\pi[0], \pi[1], \pi[2])$ and additions are modulo 3.

Theorem 34. *Suppose that $(\mathsf{PRBits.Gen}_0, \mathsf{PRBits.Gen}_1, \mathsf{PRBits.Expand})$ is an $\mathcal{F}$-programmable PCG for ε-biased bits for the function class $\mathcal{F} = \{\mathsf{id}, (x \mapsto -x)\}$, for some $\varepsilon \in [0, 1/2]$. Then, the construction in Figure 5 is a PCG for $\mathcal{D}_\varepsilon$-biased, three-party permutation correlations, where $\mathcal{D}_\varepsilon$ is the distribution on S_3 given above.*

Proof. We argue each property in turn:

Correctness. Let $(\mathsf{k}_0, \mathsf{k}_1, \mathsf{k}_2)$ be as output by $\mathsf{PCG.Gen}(1^\lambda)$, and $R_i \leftarrow \mathsf{PCG.Expand}(i, \mathsf{k}_i)$, i.e. define $R'_i = \mathsf{PRBits.Expand}(i, \mathsf{k}_i)$ for $i \in \{0, 1, 2\}$ and then, we have $R_0 = R'_0$, $R_1 = (2, \ldots, 2) + R'_1$, $R_2 = (1, \ldots, 1) + R'_2$.

Now, the R'_i denote the original subtractive shares of the bit vectors as output by $\mathsf{PRBits.Expand}(i, \mathsf{k}_i)$. Then, by construction, $R'_0 - R'_1 = (\mu_1, \ldots, \mu_\ell)$ forms a vector of ε-biased bits, and $R'_0 - R'_2 = (-\mu_1, \ldots, -\mu_\ell)$, by $\mathcal{F}$-programmability (cf. Definition 30). Let $k \in [\ell]$, then $R'_1[k] = R'_0[k] - \mu_k$ and $R'_2[k] = R'_0[k] + \mu_k$. Note that $R_0[k] + R_1[k] + R_2[k] = R'_0[k] + R'_1[k] + R'_2[k] = R'_0[k] + (R'_0[k] - \mu_k) + (R'_0[k] + \mu_k) = 0$. Hence, they form an additive sharing of zero for each k. Also, observe that $R_0[k], R_1[k], R_2[k]$ are pairwise distinct, hence they form a permutation.

Using the pseudorandomness of the PCG output, R_0 is a fresh random share in $\mathbb{F}_3$, and hence, R_0 is computationally close to uniform over $\mathbb{F}_3$, as in $\mathcal{D}_\varepsilon$. Then, the two possibilities for R_1 are exactly determined by the ε-biased bit as generated by PRBits. This shows that it is indistinguishable from a sample of $\mathcal{D}_\varepsilon$ for each $k \in [\ell]$.

Security.

We prove security via a sequence of hybrid arguments.

Hybrid $\mathcal{H}_0$. This hybrid corresponds to the $(\mathsf{k}_i, (R_j)_{j \neq i})$ as computed in Figure 5.

Hybrid $\mathcal{H}_1$. In this hybrid, we reverse sample R'_j for $j \neq i$ from R'_i and compute $(R_j)_{j \neq i}$ as computed in Figure 5.

$\mathcal{H}_1 \approx_c \mathcal{H}_0$ by the security of the PRBits PCG.

Hybrid $\mathcal{H}_2$. In this hybrid, we change how the outputs $(R_j)_{j \neq i}$ are computed. In particular, we first sample independent ε-biased bits $(\mu_1, \ldots, \mu_\ell)$ from $\mathsf{Bernoulli}(1/2 + \varepsilon)$. Then, we proceed in one of

three cases: (1) if $i = 0$ we let $R_1 := R'_0 + 2 - (\mu_1, \ldots, \mu_\ell)$ and let $R_2 := R'_0 + 1 + (\mu_1, \ldots, \mu_\ell)$; (2) if $i = 1$ we let $R_0 := R'_1 - 2 + (\mu_1, \ldots, \mu_\ell)$ and then let $R_2 := R_0 + 1 + (\mu_1, \ldots, \mu_\ell)$; or (3) if $i = 2$ we let $R_0 := R'_2 - 1 - (\mu_1, \ldots, \mu_\ell)$ and then let $R_1 := R_0 + 2 - (\mu_1, \ldots, \mu_\ell)$.

$\mathcal{H}_2 \approx_c \mathcal{H}_1$ by the pseudorandomness property of the PRBits PCG.

Hybrid $\mathcal{H}_3$. In this hybrid, we reverse sample $(R_j)_{j \neq i}$.

$\mathcal{H}_3 \approx_c \mathcal{H}_2$. In particular, note that in $\mathcal{H}_2$, the keys k_j's for $j \neq i$ are no longer used to compute the outputs. Therefore, by the correctness of the reverse sampling algorithm, the two hybrids are identical.

The argument above treats a singleton corrupted set. For two corrupted parties, their two permutation outputs uniquely determine the remaining output, so reverse sampling fixes the honest party's output and reveals no additional information beyond the ideal correlation. If all three parties are corrupted, there are no honest outputs to simulate. Thus the construction satisfies the PCG security definition for every corrupted set; as in the PCF construction, one corruption is the strongest case in which privacy about the remaining permutation is non-trivial.

■

PCG for permutation correlations over $\mathbb{F}_3$

$\mathsf{Gen}(1^\lambda)$

1 : $\mathsf{k}_0 \leftarrow \mathsf{PRBits.Gen}_0(1^\lambda)$
2 : $\mathsf{seed} \xleftarrow{\text{R}} \{0,1\}^\lambda$
3 : $\mathsf{k}_1 \leftarrow \mathsf{PRBits.Gen}_1(\mathsf{k}_0, \mathsf{seed}, f_1 = \mathsf{id})$
4 : $\mathsf{k}_2 \leftarrow \mathsf{PRBits.Gen}_1(\mathsf{k}_0, \mathsf{seed}, f_2 = (x \mapsto -x))$
5 : **return** $(\mathsf{k}_0, \mathsf{k}_1, \mathsf{k}_2)$

$\mathsf{Expand}(i, \mathsf{k}_i)$

1 : $R'_i = \mathsf{PRBits.Expand}(i, \mathsf{k}_i)$
2 : **if** $i = 0$: **return** $R_i := R'_i$
3 : **if** $i = 1$: **return** $R_i := (2, \ldots, 2) + R'_i$
4 : **else return** $R_i := (1, \ldots, 1) + R'_i$

Fig. 5: PCG for permutation correlations over $\mathbb{F}_3$

B.4 Instantiating the (1/6)-Biased Bits PCG from Quasi-Abelian Syndrome Decoding

We now show how to instantiate the previously described template, by giving a construction of a programmable PCG for ε-biased bits, where $\varepsilon = \frac{1}{6}$, based on the hardness of the quasi-abelian syndrome decoding problem [14]. As discussed above, standard LPN-based PCGs already give a generic feasibility route; the purpose of the construction in this subsection is the more concrete QA-SD-based instantiation of our programmable template.

B.4.1 QA-SD Background. The main motivation for quasi-abelian syndrome decoding is to obtain a syndrome decoding problem over a ring structure $\mathcal{R}$, where $\mathcal{R}$ is isomorphic to m copies of $\mathbb{F}_q$ for some large m. Using a cyclotomic polynomial ring $\mathcal{R} = \mathbb{F}_q[X]/f(X)$ as in ring-LWE or ring-LPN, this can be done only for $q = \Omega(m)$. Instead, quasi-abelian syndrome decoding allows us to use any $q \geq 3$.

Let $\mathbb{G}$ be a finite abelian group of order N. We define a ring $\mathcal{R}$ as the group algebra of $\mathbb{G}$ with coefficients in the finite field $\mathbb{F}_q$, that is, the set $\mathbb{F}_q[\mathbb{G}]$ of formal linear combinations:

$$\mathbb{F}_q[\mathbb{G}] := \left\{ \sum_{g \in \mathbb{G}} a_g g \mid a_g \in \mathbb{F}_q \right\}$$

Ring multiplication is defined by the convolution:

$$\left(\sum_{g \in \mathbb{G}} a_g g \right) \left(\sum_{g \in \mathbb{G}} b_g g \right) := \sum_{g \in \mathbb{G}} \left(\sum_{h \in \mathbb{G}} a_h b_{h^{-1} g} \right) g$$

The group algebra $\mathbb{F}_q[\mathbb{G}]$ is isomorphic to the vector space $\mathbb{F}_q^N$ via the map $\phi : \sum_{i=0}^{N-1} a_i g_i \mapsto (a_0, \ldots, a_{N-1})$. This allows us to define the *weight* of an element $a \in \mathbb{F}_q[\mathbb{G}]$ as the Hamming weight of $\phi(a)$. (Note that the weight is independent of the ordering of the group $\mathbb{G}$ chosen in the definition of ϕ.)

Definition 35 (Decisional quasi-abelian syndrome decoding). *Given a target weight t and compression paramater c, let $\mathcal{D}_t(\mathbb{F}_q[\mathbb{G}])$ be a noise distribution that samples weight-t elements of $\mathbb{F}_q[\mathbb{G}]$. The decisional $\mathsf{QASD}_{t,c}$ problem for $\mathcal{D}_t$ is to distinguish between the following two distributions with noticeable advantage:*

- $(\mathbf{a}', u)$, *where* $\mathbf{a}' \xleftarrow{\$} \mathbb{F}_q[\mathbb{G}]^{c-1}, u \xleftarrow{\$} \mathbb{F}_q[\mathbb{G}]$
- $(\mathbf{a}', \langle \mathbf{a}, \mathbf{e} \rangle)$, *where* $\mathbf{a} = (1, \mathbf{a}')$ *and* $\mathbf{a}' \xleftarrow{\$} \mathbb{F}_q[\mathbb{G}]^{c-1}$, $\mathbf{e} \xleftarrow{\$} \mathcal{D}_t(\mathbb{F}_q[\mathbb{G}])^c$

As in prior works [14], to improve efficiency we can use a *regular* noise distribution where each weight-t vector is divided into N/t consecutive blocks of weight-1.

Instantiating the group algebra. Following [14], we can instantiate $\mathbb{G}$ with a direct product of cyclic groups $\prod_{i=1}^{N} \mathbb{Z}/(q-1)\mathbb{Z}$. This implies that $\mathbb{F}_q[\mathbb{G}]$ is isomorphic to $\mathbb{F}_q[X_1, \ldots, X_N]/(X_1^{q-1} - 1, \ldots, X_N^{q-1} - 1)$ via an efficiently-computable isomorphism, and this is isomorphic to $\prod_{i=1}^{(q-1)^N} \mathbb{F}_q$.

Let $\mathsf{QA\text{-}Eval}_{q,N} : \mathbb{F}_q[\mathbb{G}] \to \mathbb{F}_q^{(q-1)^N}$ be the map takes an element of the group algebra, viewed as a multivariate polynomial in $\mathbb{F}_q[X_1, \ldots, X_N]/(X_1^{q-1} - 1, \ldots, X_N^{q-1} - 1)$, and evaluates it at every non-zero point. That is, $\mathsf{QA\text{-}Eval}_{q,N}(f) = (f(x_1, \ldots, x_N))_{(x_1,\ldots,x_N) \in \{1,\ldots,q-1\}^N}$. Since $\mathsf{QA\text{-}Eval}$ is a ring homomorphism, for any $f, g, h \in \mathbb{F}_q[\mathbb{G}]$, it holds that

$$\mathsf{QA\text{-}Eval}_{q,N}(fg + h) = \mathsf{QA\text{-}Eval}(f) * \mathsf{QA\text{-}Eval}(g) + \mathsf{QA\text{-}Eval}(h)$$

where $*$ denotes the entry-wise product. This allows us to directly convert additive shares of a product over $\mathbb{F}_q[\mathbb{G}]$ into a vector of shares of $(q-1)^N$ products over $\mathbb{F}_q$.

B.4.2 Construction. We want to give the parties shares of a random square $x^2 \in \mathbb{F}_3$. By choosing $x = \langle \mathbf{a}, \mathbf{e} \rangle$, as in the QASD assumption, then we can distribute shares of x^2 to two parties by giving out shares of the tensor product vector $\mathbf{e} \otimes \mathbf{e}$. Since each $e_i \in \mathbb{F}_q[\mathbb{G}]$ is t-sparse, the product $e_i e_j$ is at most t^2-sparse, and can be shared using FSS for a sum of t^2 point functions. We need to also achieve the $\mathcal{F}$-programmability feature, so that instead of shares of x^2, we may additionally distributed shares of $f(x^2)$ for some function $f \in \mathcal{F}$. We will allow $\mathcal{F}$ to be the class of linear maps from $\mathbb{F}_q \to \mathbb{F}_q$ (applied to each coordinate of x^2 independently, viewing the group algebra as a vector space over $\mathbb{F}_q$). To do this, it suffices to tweak the non-zero values of each tensored noise terms $e_i e_j$ (denoted $\mathbf{b}^i \otimes \mathbf{b}^j$ in Figure 6), applying f to each entry. Since the shares of x^2 obtained in the Expand algorithm are computed as a linear function of the shares of the tensor product, this results in the same linear map being applied to x^2.

Theorem 36. *Assume that* SPFSS *is a secure* FSS *scheme for sums of point functions and that the* $\mathsf{QA\text{-}SD}$ *assumption holds. Then, the protocol described in Figure 6 is an $\mathcal{F}$-programmable PCG for $1/6$-biased bits over $\mathbb{F}_3$, for the class of linear maps $\mathcal{F}$ described above.*

Construction PRBits

Parameters: Noise weight $t = t(\lambda)$, compression factor c, field $\mathbb{F}_3$.
A programmable SPFSS scheme $\mathsf{SPFSS} = (\mathsf{SPFSS.Gen}_0, \mathsf{SPFSS.Gen}_1, \mathsf{SPFSS.Eval})$

Public Input: $c-1$ vectors over $\mathbb{F}_3$, corresponding to the output of $\mathsf{Eval}(a_i)$ for uniformly random $\mathbf{a} = (1, a_1, \ldots, a_{c-1})$.

$\mathsf{PRBits.Gen}_0(1^\lambda)$

1 : **foreach** $i, j \in [0, c)$:
2 : $\quad K_0^{i,j} \xleftarrow{\$} \mathsf{SPFSS.Gen}_0(1^\lambda, \mathbb{G}, \mathbb{F}_3)$
3 : Let $\mathsf{k}_0 = \left((K_0^{i,j})_{i,j\in[0,c)}\right)$
4 : Output k_0

$\mathsf{PRBits.Gen}_1(\mathsf{k}_0, \mathsf{seed}, f)$

1 : Parse $\mathsf{k}_0 = \left((K_0^{i,j})_{i,j\in[0,c)}\right)$
2 : **foreach** $i \in [0, c)$:
3 : $\quad \mathbf{A}^i \xleftarrow{\$} (g_1, \ldots, g_t)_{g_i \in \mathbb{G}}$,
4 : $\quad \mathbf{b}^i \xleftarrow{\$} (\mathbb{F}_3^*)^t$ // Using randomness from seed
5 : **foreach** $i, j \in [0, c)$:
6 : $\quad K_1^{i,j} \leftarrow \mathsf{SPFSS.Gen}_1\left(K_0^{i,j}, f_{\mathbf{A}^i \otimes \mathbf{A}^j,\, f(\mathbf{b}^i \otimes \mathbf{b}^j)}\right)$
7 : Let $\mathsf{k}_1 = \left((K_1^{i,j})_{i,j\in[0,c)}\right)$
8 : Output k_1

$\mathsf{PRBits.Expand}(\sigma, \mathsf{k}_\sigma)$

1 : For $i, j \in [0, c)$:
2 : $\quad u_{\sigma, i+cj} \leftarrow \mathsf{SPFSS.Eval}(\sigma, K_\sigma^{i,j})$
3 : $z_\sigma \leftarrow \langle \mathbf{a} \otimes \mathbf{a}, \mathbf{u}_\sigma \rangle$
4 : Output $\mathsf{QA\text{-}Eval}_{3,N}(z_\sigma)$

Fig. 6: PCG for biased bits over $\mathbb{F}_3$

Proof. We prove each property in turn.

Correctness. We prove correctness via a hybrid argument.

Let $e^i = \sum_{j\in[0,t)} \mathbf{b}^i[j]\mathbf{A}^i[j]$.

Hybrid $\mathcal{H}_0$. In this hybrid, we define $z_0 = \langle \mathbf{a}\otimes\mathbf{a}, \mathbf{u}_0\rangle, z_1 = \langle \mathbf{a}\otimes\mathbf{a}, \mathbf{u}_1\rangle$ as computed in the real protocol execution and output (z_0, z_1).

Hybrid $\mathcal{H}_1$. In this hybrid, instead of computing $\mathbf{u}_1$ with $\mathsf{SPFSS.Eval}$, we replace $\mathbf{u}_1$ with $\mathbf{u}_0 - \mathbf{e}\otimes\mathbf{e}$ and compute $z_1 \leftarrow \langle \mathbf{a}\otimes\mathbf{a}, \mathbf{u}_0 - \mathbf{e}\otimes\mathbf{e}\rangle$

$\mathcal{H}_1 \approx_c \mathcal{H}_0$ by the SPFSS correctness property.

Hybrid $\mathcal{H}_2$. In this hybrid, we replace $\mathbf{u}_0$ with a randomly sampled $\tilde{\mathbf{u}}_0$. Compute $\mathbf{u}_1$ and z_0, z_1 as before using the new $\tilde{\mathbf{u}}_0$. We have $\mathbf{u}_1 = \tilde{\mathbf{u}}_0 - \mathbf{e}\otimes\mathbf{e}$, $z_0 = \langle \mathbf{a}\otimes\mathbf{a}, \tilde{\mathbf{u}}_0\rangle$, and $z_1 = \langle \mathbf{a}\otimes\mathbf{a}, \tilde{\mathbf{u}}_0 - \mathbf{e}\otimes\mathbf{e}\rangle$

$\mathcal{H}_2 \approx_c \mathcal{H}_1$ by the pseudorandomness of the outputs of $\mathsf{FSS.Eval}$ (cf. Remark 5).

Hybrid $\mathcal{H}_3$. Here, we compute the output z_1 as $z_0 - \langle \mathbf{a}, \mathbf{e}\rangle^2$. This output is identically distributed to the previous hybrid, since in $\mathcal{H}_2$ we have:

$$z_1 = \langle \mathbf{a}\otimes\mathbf{a}, \tilde{\mathbf{u}}_0\rangle - \langle \mathbf{a}\otimes\mathbf{a}, \mathbf{e}\otimes\mathbf{e}\rangle = z_0 - \langle \mathbf{a}, \mathbf{e}\rangle^2$$

Hybrid $\mathcal{H}_4$. Here, we sample $z_0, x \xleftarrow{\$} \mathbb{F}_q[\mathbb{G}]$ uniformly at random and compute $z_1 = z_0 - x^2$.

We have $\mathcal{H}_4 \approx_c \mathcal{H}_3$, due to the QASD assumption which says that $x = \langle \mathbf{a}, \mathbf{e}\rangle$ is indistinguishable from random.

Finally, we observe that this distribution is equivalent to a sample from the ideal $\frac{1}{6}$-biased bits correlation generator, as each $\mathbb{F}_3$ component of x^2 is zero with probability $\frac{1}{3}$ and one with probability $\frac{2}{3}$.

Security. Security is argued as a sequence of hybrid arguments. We use $\sigma = 1$ and note that the case for $\sigma = 0$ is symmetric. We want to show that:

$$\{(\mathsf{k}_1, x, z_0)\} \approx_c \left\{ (\mathsf{k}_1, \tilde{x}, \tilde{z}_0) \;\middle|\; \begin{array}{r} \tilde{x} \xleftarrow{\$} \mathbb{F}_3 \\ \tilde{z}_0 = \tilde{x}^2 + z_1 \end{array} \right\}.$$

Hybrid $\mathcal{H}_0$. This hybrid corresponds to the (k_1, x, z_0) computed as in the real protocol.

Hybrid $\mathcal{H}_1$. In this hybrid, we replace all the FSS keys K_1^{ij} in k_1 with simulated keys.

$\mathcal{H}_0 \approx_c \mathcal{H}_1$ via a straightforward hybrid argument replacing each of the FSS keys one-by-one with a simulated key, and relying on the correctness and security of the FSS scheme.

Hybrid $\mathcal{H}_2$. In this hybrid, we sample $\tilde{x}$ at random instead of computing is using the FSS key K_1^{ij}.

$\mathcal{H}_1 \approx_c \mathcal{H}_2$ by the $\mathsf{QA\text{-}SD}$ assumption.

Hybrid $\mathcal{H}_3$. In this hybrid, we undo the changes made in $\mathcal{H}_1$ and switch back to the original FSS keys.

$\mathcal{H}_3 \approx_c \mathcal{H}_2$ via a straightforward hybrid argument replacing each of the FSS keys one-by-one, and relying on the correctness and security of the FSS scheme.

At this point, we have that $\mathcal{H}_3$ is distributed exactly as $(\mathsf{k}_1, \tilde{x}, \tilde{z_0})$, concluding the proof.

Programmability. The construction is programmable (that is, we have $z_0 - z_2 = f(z_0 - z_1)$) due to the programmability of the FSS. In particular, this follows from the fact that by scaling the non-zero values for the point functions defined by $\mathbf{b}^i \otimes \mathbf{b}^i$ by some constant $\Delta \in \mathbb{F}_q$ at key generation time, the secret shared output becomes a share of $\Delta \cdot x^2$ instead of x^2. The programmability of the FSS ensures that we then obtain the necessary $\mathcal{F}$-programmability feature, since we can keep the share z_0 independent (generated by Gen_0) and get $z_0 - z_2 = \Delta(z_0 - z_1)$. This immediately generalizes to linear functions f.

In addition, as was discussed earlier, for random $x \in \mathbb{F}_3$, x^2 is 1 for $x = 1, 2$ (2/3 of the times) and 0 for $x = 0$ (1/3 of the times). This means that x^2 is distributed according to $\mathsf{Bernoulli}(1/2{+}1/6)$ over $\mathbb{F}_3$ and so our construction is a 1/6-biased bits correlation generator over $\mathbb{F}_3$.

■

B.5 Implementation and parameters

We implement the biased PCG construction in C by adapting the open-source implementation of the FOLEAGE [13] PCG for programmable OLE correlations. In particular, we modify their implementation by adapting it to work over $\mathbb{F}_3$ and use their parameter estimation script to derive a new set of parameters tailored to our construction. The parameters c (compression factor) and t (noise weight) influence the size of the PCG key and evaluation time. We set $c = 5$ and $t = 16$, which requires evaluating $(ct)^2 = 6400$ DPFs to compute PCG.Expand.

The choice of $(c = 5, t = 16)$ corresponds to a conservative parameter choice based on the parameter estimation script of [13]. We then replace their use of DPFs with programmable DPFs, to match Figure 6. In particular, we use $T = 40$, which gives us correctness $1 - 2^{-40}$.

Our programmable DPF implementation takes advantage of the `AES-NI` instruction to implement a fast PRG G using fixed-key AES (from the OpenSSL library [72]) and the Davies-Meyer transform. However, programmable DPFs do not allow for the optimizations exploited in the implementation of the FOLEAGE PCG [13] and are overall concretely more expensive to evaluate. In particular, the SPFSS (sum of many DPFs) evaluation accounts for 98% of the total computation time of the PCG expansion.

B.5.1 Recent attack on QA-SD parameters Very recently, a new attack has been demonstrated on the QA-SD assumption, which uses sparse multivariate interpolation algorithms to break various parameter sets that were proposed in practice [16]. We caution that our implementation, which uses parameters based on [13], is also vulnerable to the attack, and so insecure. To resist the attack, it would suffice to increase the parameter c from 5 to 8. While we have not yet been able to run our benchmarks with this change, we estimate it would incur roughly a factor-4 slowdown on the current results.

D	PCG.Expand	**Key Size** (per party)
2^{18}	4 minutes	0.71 MB
2^{20}	14 minutes	0.77 MB
2^{22}	48 minutes	0.84 MB

Table 2: Performance of our PCG implementation on a `c5.metal` EC2 instance. We set the noise parameter to $t = 16$ and set the compression factor $c = 5$, following the parameter selection script of [13].

B.6 Benchmarks

We run a set of benchmarks using AWS `c5.metal` (3.4GHz CPU). All experiments are averaged across ten trials and evaluated on a single core. We report the benchmark results in Table 2. The parameter $D = 2^n$ determines the number of bits we generate in total. Our choice of DPF range 2^{18}, 2^{20}, and 2^{22} correspond to the size of a regular noise block when $D = 2^{22}$, $D = 3^{24}$, and $D = 3^{26}$, respectively. Evaluating the PCG requires $(ct)^2$ calls to the DPF on domain size D/t (using the regular noise optimization described in [13]). The DPF evaluation cost ends up being the dominant factor (approximately 97% of the total computation). However, we obtain an amortized performance of approximately 1,200 permutations per second.

B.7 Performance Estimate for the Unbiased Construction

For the HSS-based construction from Section 4, we additionally give some performance estimates when using an HSS scheme based on Paillier, and an HSS-friendly (weak) PRF.

Instantiating the HSS. We consider the two-party HSS construction from [2], instantiated with ElGamal encryption in the Paillier group $\mathbb{Z}_{N^2}$ for a 2048-bit RSA modulus N. The secret key is a short ($\geq 2\lambda$-bit) integer exponent. Using a recent result of [5], this yields an HSS whose security reduces to DCR together with the hardness of the short-exponent discrete logarithm assumption. With this construction, a single ElGamal ciphertext has size $4 \log N$ bits, and each input to the HSS requires 2 ciphertexts, for a total of $8 \log N$ bits per input. We assume a setup procedure that has distributed shares of the secret key between the two parties. After this setup, the HSS supports homomorphic evaluation of arbitrary, *restricted-multiplication* circuits on the inputs — namely, arithmetic circuits where every multiplication gate is of fan-in two and includes at least one circuit input wire as an input to the gate.

Instantiating the HSS-friendly PRF. We instantiate the pseudorandom function $(F.\mathsf{KeyGen}, F.\mathsf{Eval})$ using a Goldreich-style weak-PRF [7] of low locality $\ell = O(\log \lambda)$, given by:

$$F_s(x) = P(s_{i_1}, \ldots, s_{i_\ell})$$

Here, the indices i_j are randomly determined based on the input x (in practice, using, for instance, a hash function modeled as a random oracle), while the predicate P is a fixed, locality-ℓ function of the PRF key s.

To instantiate P, one way to ensure a relatively high stretch is to use the XOR-majority predicate $P(x) = x_1 \oplus \cdots \oplus x_{\ell_1} \oplus \mathsf{MAJ}(x_{\ell_1+1}, \ldots, x_{\ell_1+\ell_2})$, where $\ell = \ell_1 + \ell_2$. Although this predicate was recently shown to be susceptible to bit-fixing correlation attacks [52], this can be mitigated by increasing the seed size until the attacks are no longer feasible. Concretely, we consider two instantiations:

- $\ell_1 = 32, \ell_2 = 64$ from [42], with a seed of length 512 (increased from 256 in [42]) and up to 2^{40} pseudorandom output bits.
- $\ell_1 = 5, \ell_2 = 15$ from [39], with a seed length of 2048 and up to 2^{33} output bits.

Both parameter sets should resist the attacks from [52].

To evaluate the PRF as a restricted-multiplication circuit, first, note that since the majority function on k inputs is symmetric, it can be expressed as a degree-k polynomial over any finite field $\mathbf{F}_p$ for $p > k$, by summing the input to obtain a value in $\{0, \ldots, k\}$ and interpolating. Therefore, majority can be computed using $k - 1$ multiplications and $O(k)$ additions/scalar multiplications, where in our case, $k = \ell_2$. The XOR part of the PRF can be computed with a chain of $\ell_1 + 1$ multiplication gates, giving a total of $\ell = \ell_1 + \ell_2$ multiplications.

Efficiency. The storage and communication cost for each party consists of $8 \log N$ bits per input to store the HSS ciphertexts. With the first parameter set, we get a total seed size of ≈ 1050 kB per party, and with the second, ≈ 4200 kB. The dominant computational cost of evaluating the PRF inside HSS is the multiplication gates. Each multiplication consists of a distributed discrete logarithm procedure, whose dominant cost can be approximated by 1 exponentiation in $\mathbb{Z}_{N^2}$ with an $\approx 3\lambda$-bit exponent. Based on estimates for exponentiations from a modern laptop with an Intel i7-1365U CPU, this should cost around 1ms. This gives a total evaluation cost of around 64ms per permutation output, for the first parameter set, or only 15ms for the second. This seems to be a reasonable latency for our applications to single-secret leader election and anonymous broadcast.