Efficient Pseudorandom Correlation Generators for Any Finite Field

Zhe Li, Chaoping Xing, Yizhou Yao, Chen Yuan

School of Electronic Information and Electrical Engineering,
Shanghai Jiao Tong University, Shanghai, China
lizh0048@e.ntu.edu.sg,{xingcp,yaoyizhou0620,chen_yuan}@sjtu.edu.cn

Abstract. Correlated randomness lies at the core of efficient modern secure multi-party computation (MPC) protocols. Costs of generating such correlated randomness required for the MPC online phase protocol often constitute a bottleneck in the overall protocol. A recent paradigm of *pseudorandom correlation generator* (PCG) initiated by Boyle et al. (CCS'18, Crypto'19) offers an appealing solution to this issue. In sketch, each party is given a short PCG seed, which can be locally expanded into long correlated strings, satisfying the target correlation. Among various types of correlations, there is oblivious linear evaluation (OLE), a fundamental and useful primitive for typical MPC protocols on arithmetic circuits. Towards efficient generating a great amount of OLE, and applications to MPC protocols, we establish the following results:
(i) We propose a novel *programmable* PCG construction for OLE over any field $\mathbb{F}_p$. For kN OLE correlations, we require $O(k \log N)$ communication and $O(k^2 N \log N)$ computation, where k is an arbitrary integer ≥ 2. Previous works either have quadratic computation (Boyle et al. Crypto'19), or can only support fields of size larger than 2 (Bombar et al. Crypto'23).
(ii) We extend the above OLE construction to provide various types of correlations for any finite field. One of the fascinating applications is an efficient PCG for two-party *authenticated Boolean multiplication triples*. For kN authenticated triples, we offer PCGs with seed size of $O(k^2 \log N)$ bits. To our best knowledge, such correlation has not been realized with sublinear communication and quasi-linear computation ever before.
(iii) In addition, the *programmability* admits efficient PCGs for multi-party Boolean triples, and thus the first efficient MPC protocol for Boolean circuits with *silent* preprocessing. In particular, we show kN m-party Boolean multiplication triples can be generated in $O(m^2 k \log N)$-bit communication, while the state-of-the-art FOLEAGE (Asiacrypt'24) requires a broadcast channel and takes $mkN + O(m^2 \log kN)$ bits communication.
(iv) Finally, we present efficient PCGs for circuit-dependent preprocessing, matrix multiplications triples, and string OTs etc. Compared to previous works, each has its own right.

Table of Contents

1 Introduction

Correlated randomness stands as a fundamental and crucial part in secure multi-party computation (MPC) protocols with preprocessing. Thanks to the celebrated work of Beaver [Bea91], studies of concretely efficient MPC are maturing, and various settings are explored, e.g., honest majority [EGPS22], dishonest majority [DPSZ12, KOS16, EGP+23], and fluid MPC [RS22]. All these modern MPC protocols are designed in a preprocessing model, in which parties are given access to a sufficient number of correlated random strings, typically $\Omega(N)$ for securely computing circuits with N gates.

Many types of correlations have been used for wide scenarios: for instance, oblivious linear function evaluation (OLE) enables (semi-honest) secure 2-party computation (2-PC) on arithmetic circuits, and it also allows for m-party (semi-honest) secure computation on arithmetic circuits, through building m-party Beaver triples, which further admits malicious secure MPC via constructing authenticated Beaver triples. Oblivious transfer (OT) enables semi-honest secure 2-PC on Boolean circuits by Yao's garbled circuit technique [Yao86], and can be used to generate various types of correlations, including OLE. Finally, matrix multiplication triples become useful recently, receiving attentions from industry in privacy-preserving machine learning.

Hence, there is a great demand for concretely efficient approaches to generating various and numerous correlated randomness, so that MPC can be applied to more real-life scenarios, wider and larger. Pseudorandom correlation generators (PCGs) introduced in [BCGI18, BCG+19a, BCG+19b] and the subsequent works [BCG+19a, YWL+20, BCG+20, BCG+22, BCG+23, RRT23, BCCD23, ABG+24] offer an appealing solution to *efficient* preprocessing. Informally, in a PCG-based preprocessing, each party is given a *short* PCG seed, who then can *locally* expand their own seed and get the target long correlations. Such procedure is called *silent preprocessing*, since no interaction is required as long as PCG seeds are distributed.

Technically, most of existing PCG constructions have an "extension" flavor. The PCG seeds essentially specify short correlations, then the seeds are locally expanded via variants of learning parity with noise (LPN) assumptions [BFKL93] or syndrome decoding assumptions [BCG+22, BCCD23, RRT23]. Depending on the correlations and settings, generating of seeds relies on efficient function secret sharing schemes (FSS) [GI14, BGI15, BGI16], and distributed setup protocols with semi-honest/malicious security [BCG+20, GYW+23, ZGY+24, BBC+24]. In this work, we start with PCG constructions for OLEs, which is a critical building block for other correlations and applications.

PCG for OLEs. The pioneer work [BCG+19b] constructs efficient PCGs for several correlations from LPN assumptions, including OLE. Their initial construction generally works for any finite field with a sublinear seed size, but has a major downside of $\tilde{O}(N^4)$ computation for N OLE correlations. Though later it is optimized to $\tilde{O}(N^2)$ computation, which is still very expensive, as N is usually huge. The follow-up work [BCG+20] presents a new PCG with $\tilde{O}(N)$ computation from using a structured LPN assumption, namely, the Ring-LPN

assumption. Though [BCG+20] achieves good concrete efficiency, the field size is restricted to be larger than N, inherently induced by Ring-LPN. Following the routine from Ring-LPN, Bombar et al. [BCCD23] put a step forward, achieving OLE over arbitrary fields only except for the significantly interesting case of $\mathbb{F}_2$. Their results are obtained by introducing the quasi-abelian syndrome decoding (QA-SD) assumption, intuitively a "multi-variate" sense of Ring-LPN.

We remark that for the interesting case of $\mathbb{F}_2$, where OLE over $\mathbb{F}_2$ is actually equivalent to bit OT, there do exist efficient PCG constructions [BCG+19a, BCG+19b, YWL+20, BCG+23]. However, these PCGs for OTs are obtained by hashing random Vector-OLE (VOLE) correlations (from PCG), hence they are not easy to be extended to PCGs for other correlations, for example, Boolean multiplication triples. Moreover, the involved hash function also prevents these PCGs from being extended to multi-party correlations, while other PCGs [BCG+20, BCCD23] without hashing likely enjoy a "programmability" property, allowing for the multi-party generalization.

The task of constructing efficient PCGs for authenticated triples is much more challenging, even in the 2-party case. Because authenticated triples are actually degree-3 relations while OLEs are of degree-2. Though the Ring-LPN/QA-SD based approaches from [BCG+20, BCCD23] imply constructions for fields with size ≥ 3, it is unclear how to construct PCGs for authenticated Boolean multiplication triples. Therefore, we ask the following question:

Do there exist efficient programmable PCGs for OLEs, and PCGs for authenticated multiplication triples over any finite field, in particular, the binary field $\mathbb{F}_2$?

1.1 Our Contributions

In this work, we revisit and generalize the approach of [BCG+20, BCCD23] for constructing PCG for OLEs, obtaining a new framework by introducing a simple but effective function, the *Trace* function for the polynomial rings defined over $\mathbb{F}_{p^k}$. This allows us to give an efficient programmable PCG for OLEs over any fields. We also implement and evaluate concrete performances of our constructions, which are as efficient as previous PCGs over large fields. We then extend our OLE construction for various useful correlations, in particular, including authenticated Boolean multiplication triples. Other useful correlations we show are multi-party Beaver triples, matrix multiplication triples, and string OTs, etc.

PCG for OLEs over any finite field. We present new PCG constructions for random OLE correlations, in which previous constructions either have large computational complexity, or have the undesirable restriction on the field size. In Table 1, we summarize our results and give comparisons with existing PCGs.

In an OLE correlation, for $\sigma \in \{0,1\}$, party P_σ holds $x_\sigma, z_\sigma \in \mathbb{F}_p$, such that $x_0 \cdot x_1 = z_0 + z_1$. In other words, each party holds an additive share of $x_0 \cdot x_1$, and we simply denote it as $[x_0 \cdot x_1]$. Our constructions build upon existing, well-studied Ring-LPN/QA-SD assumptions and function secret sharing (FSS) for sums of point functions (SPFSS). For simplicity, here we present the results from QA-SD, and the Ring-LPN based constructions are deferred to Appendix D.

As for fast computation, the generic FFT with quasi-linear complexity [Obe07, Algorithm 74] can be applied, and it is optimized for some concrete rings in [BBC+24, Section 4.3].

	N OLEs	Comm.	Comp.	Programmability	Assumption
[BCG+19b]	Any $\mathbb{F}_p$	$O(\lambda^3 \log N)$	$O(N^2 \log N)$	Yes	Dual-LPN
[BCG+20]	$\mathbb{F}_p, p > N$	$O(\lambda^3 \log N)$	$O(N \log N)$	Yes	Ring-LPN
[BCCD23]	$\mathbb{F}_p, p > 2$	$O(\lambda^3 \log N)$	$O(N \log N)$	Yes	QA-SD
[BCG+23]	$\mathbb{F}_2$	$O(\lambda^2 \log N)$	$O(N)$	No	Dual-LPN
This work	Any $\mathbb{F}_p$	$O(\lambda^3 \log N)$	$O(N \log N)$	Yes	QA-SD

Table 1. Comparisons to previous PCGs for OLE. Let λ denote the security parameter. By "Comm." we refer to the communication complexity while "Comp." stands for the computation complexity. All these approaches allow for arbitrary polynomial stretch (the ratio of number of correlations and communication cost).

Authenticated multiplication triples. Informally, authenticated multiplication triples are multiplication triples with message authentication codes (MACs). To authenticate a multiplication triple $([x], [y], [z])$ with $x, y, z \in \mathbb{F}_p, x \cdot y = z$, parties additionally holds $([x \cdot \Delta], [y \cdot \Delta], [z \cdot \Delta])$, where $\Delta \in \mathbb{F}_{p^k}$ is the global key and keeps identical among triples. We denote such a triple by $(\llbracket x \rrbracket, \llbracket y \rrbracket, \llbracket z \rrbracket)$. Note that x, y, z are additively shared over $\mathbb{F}_p$, while $\Delta \cdot x, \Delta \cdot y, \Delta \cdot z$ are additively shared over $\mathbb{F}_{p^k}$ with $k = O(\lambda / \log p)$. The construction generalizes the above PCG for OLE, and also relies on QA-SD assumptions and SPFSS schemes.

Theorem 1.1 (PCG for authenticated multiplication triples, informal). *Assuming the QA-SD assumption and the* SPFSS *in [BGI16], there exists a secure PCG construction for generating $O(N)$ authenticated multiplication triples, with seed size $O(\lambda^3 \log N)$ and computational complexity $O(N \log N)$, where λ is the security parameter.*

We remark that taking $p = 2$ in Theorem 1.1 induces extremely interesting results over $\mathbb{F}_2$, i.e., the PCG for authenticated Boolean triples. Moreover, we consider semi-honestly/maliciously secure distributed setup protocols for OLEs/triples, which rely on the malicious DPF construction in [BCG+20]. In particular, we evaluate the concrete seed expansion time on our machine following the way in [BCG+20]. The results are summarized in Table 2, showing that our constructions for authenticated Boolean triples are as efficient as those for authenticated multiplication triples over large fields.

Other Applications. Since our PCG for OLEs is programmable, it is extended to m-party semi-honest Beaver triples by a strandard transformation, leading to the seed size $O(m \log N)$. Compared to the m-party Boolean triple preprocessing of [BBC+24], our approach offers a completely silent preprocessing with *sublinear* communication $O(m^2 \log N)$, while [BBC+24] requires a broadcast channel and has communication complexity of $mN + O(m^2 \log N)$ bits. In Table 3, we report

Protocol	Field	Triple Generation	Comm.
Overdrive [KPR18]	128 bit prime field	$30,000/s$	2 GB
PCG [BCG+20] estimated	128 bit prime field	$50,000/s$	4.2MB
This work	Boolean with 128-bit MAC key	$43,000/s$	47.54MB

Table 2. The cost refers to the generation of authenticated multiplication triples for two parties. Both the Overdrive results and the estimated results from [BCG+20] are taken from [BCG+20]. The concrete performance of this work is derived from running our program for generating 3^{16} authenticated Boolean multiplication triples.

the concrete efficiency of our approach for generating multiparty Boolean triples, and compare with the state-of-the-art works.

	Party No.	Triple No.	Comm.	Broad.	Time. (s)
SoftSpokenOT [Roy22] ($k = 8$)	2	10^9	3.7 GB	0	211
$\mathbb{F}_4$OLEAGE [BBC+24, Table 1]	2	10^9	33.5 MB	0	81
$\mathbb{F}_4$OLEAGE [BBC+24, Our machine]	2	10^9	33.5 MB	0	83
This work	2	10^9	33.6 MB	0	162
SoftSpokenOT[Roy22] ($k = 8$)	10	10^9	34 GB	0	1900
$\mathbb{F}_4$OLEAGE [BBC+24, Table 1]	10	10^9	0.6 GB	0.12 GB	1463
$\mathbb{F}_4$OLEAGE [BBC+24, Our machine]	10	10^9	0.6 GB	0.12 GB	1511
This work	10	10^9	0.6 GB	0	2932

Table 3. Concrete evaluation performances on a PC with Intel(R) Xeon(R) Gold 5220R 2.20GHz CPU and 128GB of RAM, following the same way as [BBC+24]. For fairness, we run the $\mathbb{F}_4$OLEAGE source code on our machine and use the same parameters as $\mathbb{F}_4$OLEAGE: $c = 3$, $t = 27$, and $n = 16$. By "Comm.", we refer to the per-party communication over a point-to-point channel. By "Broad.", we refer to the per-party communication over a broadcast channel. By "Time.", we refer to the seed expansion time in the localhost setting.

Following the routine of [BCG+20, BCCD23], our approach also supports circuit-dependent preprocessing. For computing N copies on the circuit on the circuit $\mathcal{C}$ over $\mathbb{F}_p$ with different inputs, we offer a silent preprocessing with seed size $O(|\mathcal{C}| \log N)$. We also construct PCGs for matrix multiplication triples. Concretely, for N/k triples of $k \times k$ matrices over $\mathbb{F}_p$, we have seed size of $O(k\lambda^3 \log N)$, while the previous work [BCG+20] for large fields has seed size of $O(k^2\lambda^3 \log N/k)$[1] and the recent work [LXYZ24] for any field has seed size of $O(N\lambda^2 \log k)$. Finally, we give a PCG construction for OTs with seed size of

[1] The subsequent work [BCG+22] implicitly gives a pseudorandom correlation function (PCF) for matrix triples, which can produce an arbitrary number of correlations but the computation per correlation is much higher (2-3M PRG evaluations).

$O(\lambda^3(k+\log N))$ for generating N k-bit string OT correlations. It is worth to mention that our construction for OTs does not require correlation-robust hash functions, which is a feasible result and of independent interest.

1.2 Organization

We provide a detailed technical overview of our approach in Section 2, and preliminaries in Section 3. We give related mathematical results in Section 4 with proofs deferred to Appendix B. We give the PCG construction for OLE over any field in Section 5, and the construction for authenticated multiplication triples over any field in Section 6. In Section 7, we discuss more applications of our approach, including multi-party multiplication triples and circuit-dependent preprocessing in Section 7.1, matrix triples in Section 7.2, and finally OTs in Section 7.3. We give specific PCG setup protocols with semi-honest/malicious security in Section 8, and underlying DPF constructions in Appendix C. In Section 9, we analyze security of underlying assumptions and select appropriate parameters for our PCG constructions. In Appendix A, we give more preliminaries, and in Appendix D, we show constructions from Ring-LPN assumptions.

2 Our Techniques

2.1 Limitations of PCG for OLEs from [BCG+20, BCCD23]

We briefly review the framework of PCG for OLE over $\mathbb{F}_{p^k}$ in [BCG+20, BCCD23]. Suppose the goal is to generate $N := d^n$ random OLE correlations over $\mathbb{F}_{p^k}$, where p is a prime and $n, k \geq 1$. Let $\mathcal{R} := \mathbb{F}_{p^k}[\mathtt{X}_1, \ldots, \mathtt{X}_n]/(f(\mathtt{X}_1), \ldots, f(\mathtt{X}_n))$, where $f(\mathtt{X}) = \mathtt{X}^d - 1$, and $d \mid p^k - 1$. The framework relies on a ring isomorphism $\mathcal{R} \simeq \mathbb{F}_{p^k}^N$, which is implied by the Chinese remainder theorem (CRT), so that it suffices to distribute one OLE correlation over $\mathcal{R}$. Let $a \xleftarrow{\$} \mathcal{R}$ be the public input, and s_σ, e_σ be t-sparse elements of $\mathcal{R}$, where $\sigma \in \{0,1\}$. The Ring-LPN/QA-SD assumption[2] implies that $x_\sigma := a \cdot s_\sigma + e_\sigma$ is pseudorandom, for each $\sigma \in \{0,1\}$. The key observation is the following:

$$x_0 \cdot x_1 = (a \cdot s_0 + e_0) \cdot (a \cdot s_1 + e_1) = a^2 \cdot (s_0 s_1) + a \cdot (s_0 e_1 + s_1 e_0) + e_0 e_1.$$

Hence, it suffices to distribute additive shares of cross-terms $s_0 s_1, s_0 e_1, s_1 e_0, e_0 e_1$ to the two parties, which then can be locally converted into additive shares of $x_0 \cdot x_1$. By the isomorphism $\mathcal{R} \simeq \mathbb{F}_{p^k}^N$, additive sharings of $x_0 \cdot x_1$ over $\mathcal{R}$ are equivalent to N OLE correlations over $\mathbb{F}_{p^k}$, as desired. Since $s_0 s_1, s_0 e_1, s_1 e_0, e_0 e_1$ are essentially t^2-sparse elements of $\mathcal{R}$, sharing of them can be done efficiently from using SPFSS. Moreover, the resulting construction has programmability, admitting PCG for multi-party Beaver triples.

[2] The Ring-LPN assumption corresponds to the univariate case, where n is always 1 and $N \mid p^k - 1$. While the QA-SD assumption corresponds to the multivariate case, where $N = d^n$ and $d \mid p^k - 1$.

However, the above approach has an undesirable and inherent restriction on the field size. Although, Bombar et al. [BCCD23] significantly relax the restriction of $p^k > N$ from Ring-LPN via introducing QA-SD assumptions, they fail in the significantly interesting case of $\mathbb{F}_2$. The main reason is that $\mathbb{F}_2$ has only one invertible element, and setting $f(\mathtt{X}) := \mathtt{X}(\mathtt{X}-1)$ incurs a severe attack on the QA-SD assumption as shown in [BCCD23], while setting $f(\mathtt{X}) := \mathtt{X}-1$ leads to a trivial extension. Also, [BCCD23, Theorem 47] elaborates an impossibility result on it. We would like to mention that using advanced mathematical tools of algebraic geometry seems unhelpful, turning the problem to be more complicated [BCCD23, Appendix D].

Though the QA-SD approach does not offer a direct solution to correlations over $\mathbb{F}_2$, it is shown in FOLEAGE [BBC+24] that multi-party Boolean triples can be obtained by first generating OLEs over $\mathbb{F}_4$, and then turning into $\mathbb{F}_2$ via reconstructing parts of shares. However, this approach requires additional interaction and linear communication for sharing reconstruction, hence not a *silent* preprocessing. Moreover, it is not clear how to further efficiently construct silent PCG for authenticated Boolean triples from such $\mathbb{F}_4$ OLEs.

2.2 Trace to the Rescue

To overcome the barrier of the field size restriction, a fundamentally different approach is in demand. Instead of directly building PCG constructions upon assumptions on polynomial rings or algebraic curves over $\mathbb{F}_2$, we resort to finding an approach that allows to convert $\mathbb{F}_{2^k}$ correlations into desired $\mathbb{F}_2$ correlations *efficiently* and *locally*. Among maps from $\mathbb{F}_{2^k}$ to $\mathbb{F}_2$, the trace map seems a good candidate that might help, as it is additively homomorphic and well-studied. Bearing this in mind, we come up with a novel approach based on trace.

To this end, we first extend the notion of trace function from field elements of $\mathbb{F}_{p^k}$ to vectors in $\mathbb{F}_{p^k}^N$ in a natural way, and then examine the properties on the ring $\mathcal{R}$, since we have the isomorphism $\mathcal{R} \simeq \mathbb{F}_{p^k}^N$ induced by CRT. Specifically, for a vector $\mathbf{v} \in \mathbb{F}_{p^k}^N$, the trace of $\mathbf{v}$ is defined as

$$\mathsf{Tr}(\mathbf{v}) := (\mathsf{Tr}(v_1) \dots \mathsf{Tr}(v_N)) \in \mathbb{F}_p^N.$$

Recall that for an arbitrary $\alpha \in \mathbb{F}_{p^k}$, the trace of α is defined as $\mathsf{Tr}(\alpha) := \sum_{i=0}^{k-1} \alpha^{p^i} \in \mathbb{F}_p$, so we define *trace function* of an arbitrary $f \in \mathcal{R}$ as

$$\mathsf{Tr}(f) := \sum_{i=0}^{k-1} f^{p^i}.$$

We systematically study the above trace function over $\mathcal{R}$, and we refer the details to Section 4. Informally, it preserves most properties of the trace function over $\mathbb{F}_{p^k}$, for instance, $\mathsf{Tr}(\cdot)$ is $\mathbb{F}_p$-linear, i.e., $\forall \alpha \in \mathbb{F}_p, \forall f \in \mathcal{R}$, $\mathsf{Tr}(\alpha \cdot f) = \alpha \cdot \mathsf{Tr}(f)$. Moreover, let $\phi : \mathcal{R} \to \mathbb{F}_{p^k}^N$ denote the isomorphism of $\mathcal{R} \simeq \mathbb{F}_{p^k}^N$. Then the trace function for $\mathcal{R}$ has the following desired property.

$$\forall f \in \mathcal{R}, \phi(\mathsf{Tr}(f)) \in \mathbb{F}_p^N.$$

It is not hard prove that $\phi(\mathsf{Tr}(f)) = \mathsf{Tr}(\phi(f)) \in \mathbb{F}_p^N$, i.e., ϕ and Tr are "commutative". Note that here we abuse the trace function $\mathsf{Tr}(\cdot)$ over $\mathcal{R}$ and over $\mathbb{F}_{p^k}^N$. The above described properties make it possible to obtain efficient PCGs for any field.

High-level idea of our approach. Let $x_\sigma := as_\sigma + e_\sigma$, for $\sigma \in \{0,1\}$, be two QA-SD/Ring-LPN samples. Towards bypassing the field restriction, the core idea is to share the product of the traces of x_0, x_1, i.e., $\mathsf{Tr}(x_0) \cdot \mathsf{Tr}(x_1)$, rather than $x_0 \cdot x_1$ as in previous works. In more detail, as long as we can succinctly distribute $x_\sigma, z_\sigma \in \mathcal{R}$ to P_σ satisfying

$$\mathsf{Tr}(x_0) \cdot \mathsf{Tr}(x_1) = z_0 + z_1,$$

the two parties immediately obtain $[\mathsf{Tr}(x_0) \cdot \mathsf{Tr}(x_1)]_{\mathcal{R}}$. By the ring isomorphism $\mathcal{R} \simeq \mathbb{F}_{p^k}^N$, $\phi(z_0 + z_1)$ is destined to be over $\mathbb{F}_p^N$ with $\phi(z_0), \phi(z_1) \in \mathbb{F}_{p^k}^N$ rather than $\mathbb{F}_p^N$. Therefore, the $k-1$ high-dimensional arrays of $\phi(z_0), \phi(z_1) \in (\mathbb{F}_p^k)^N$ cancel out and can be discarded. Since p is the characteristic of $\mathbb{F}_{p^k}$, $\mathsf{Tr}(x_\sigma) = \sum_{i=0}^{k-1}(as_\sigma + e_\sigma)^{p^i}$ has $2k$ terms and $\mathsf{Tr}(x_0) \cdot \mathsf{Tr}(x_1) = \sum_{i,j\in[0,k-1]}(a^{p^i}s_0^{p^i} + e_1^{p^i})(a^{p^j}s_0^{p^j} + e_1^{p^j})$ has $4k^2$ terms . Then it suffices to use FSS to share the cross-terms, e.g., $s_0^{p^i}e_1^{p^j}$, leading to a PCG having seed size scaling with $4k^2$. By further utilizing algebraic properties of trace, we are able to achieve a PCG of seed size scaling with $4k$, i.e., $O(\lambda k t^2 \log N)$ for kN OLE correlations over an arbitrary field $\mathbb{F}_p$, where t is set $O(\lambda)$ for guaranteeing security of the QA-SD assumptions and k is an arbitrary integer ≥ 2.

2.3 Warm-up: from $\mathbb{F}_{2^k}$ to $\mathbb{F}_2$ via Trace

We give a warm-up construction, focusing on the special case of OLE over $\mathbb{F}_2$. Assume $\mathcal{R}_k := \mathbb{F}_{2^k}[\mathtt{X}_1 \dots \mathtt{X}_n]/(\mathtt{X}_1^{2^k-1} \dots \mathtt{X}_n^{2^k-1})$. Without loss of generality, we first set $k = 2$, $N = 3^n$, $\xi \in \mathbb{F}_4$ s.t. $\mathbb{F}_4 = \mathbb{F}_2(\xi)$. Let $a \xleftarrow{\$} \mathcal{R}_2$ be the public input, and s_σ, e_σ be two random t-sparse elements of $\mathcal{R}_2$, where $\sigma \in \{0,1\}$. Given $(a, b_\sigma = a \cdot s_\sigma + e_\sigma)$, denote QA-SD$(\mathcal{R}_2, t)$ as the problem to distinguish (a, b) from (a, u) with $u \xleftarrow{\$} \mathcal{R}_2$. The hardness of QA-SD$(\mathcal{R}_2, t)$ implies that $x_\sigma := a \cdot s_\sigma + e_\sigma$ is pseudorandom for $\sigma \in \{0,1\}$. Recall that for OLEs over $\mathbb{F}_4$ (i.e., for $\sigma \in \{0,1\}$, P_σ obtains random $x_\sigma, z_\sigma \in \mathcal{R}_2$, such that $x_0 \cdot x_1 = z_0 + z_1$), following the construction of [BCCD23], it suffices to use SPFSS to additively share $s_0s_1, s_0e_1, e_0s_1, e_0e_1$, according to the following equation:

$$x_0 \cdot x_1 = (a \cdot s_0 + e_0) \cdot (a \cdot s_1 + e_1) = a^2 \cdot (s_0s_1) + a \cdot (s_0e_1 + s_1e_0) + e_0e_1. \quad (1)$$

However, it is not clear how to *efficiently* and *locally* convert additive shares of $x_0 \cdot x_1$ over $\mathcal{R}_2 \simeq \mathbb{F}_4^N$ to OLE correlations over $\mathbb{F}_2$.

We also start with QA-SD$(\mathcal{R}_2, t)$, but instead we try to employ trace of $\mathbb{F}_4$ over $\mathbb{F}_2$ to achieve the local conversion. To this end, we embed the trace structure to Eq.(1). Our goal is to additively share $\mathsf{Tr}(\phi(x_0)) * \mathsf{Tr}(\phi(x_1)) \in \mathbb{F}_2^N$, which is

equivalent to share $\mathsf{Tr}(x_0) \cdot \mathsf{Tr}(x_1) \in \mathcal{R}_2$. We have the following observation:

$$\begin{aligned}
&\mathsf{Tr}(x_0) \cdot \mathsf{Tr}(x_1) = (x_0^2 + x_0)(x_1^2 + x_1) = \prod_{\sigma \in \{0,1\}} \left((a \cdot s_\sigma + e_\sigma)^2 + a \cdot s_\sigma + e_\sigma\right) \\
&= \prod_{\sigma \in \{0,1\}} (a^2 \cdot s_\sigma^2 + a \cdot s_\sigma + e_\sigma^2 + e_\sigma) \\
&= a^4 \cdot s_0^2 s_1^2 + a^3 \cdot (s_0^2 s_1 + s_0 s_1^2) + a^2 \cdot (s_0^2 e_1^2 + s_0^2 e_1 + s_0 s_1 + s_1^2 e_0^2 + s_1^2 e_0) \\
&+ a \cdot (s_0 e_1^2 + s_0 e_1 + s_1 e_0^2 + s_1 e_0) + (e_0^2 e_1^2 + e_0^2 e_1 + e_0 e_1^2 + e_0 e_1).
\end{aligned} \tag{2}$$

It is easy to see that the above formula on the right has $16 = 4 \cdot 4$ terms, and it suffices to use SPFSS to distribute additive shares of these values. Suppose each term is shared as (α_i, β_i) for $i \in [1, 16]$, i.e., party $\mathrm{P}_0, \mathrm{P}_1$ obtains $\{\alpha_i\}_i, \{\beta_i\}_i$, respectively, such that $\mathsf{Tr}(x_0) \cdot \mathsf{Tr}(x_1) = \sum_{i=1}^{16} (\alpha_i + \beta_i)$. Then P_0 and P_1 can locally compute $\boldsymbol{\alpha} = \phi(\sum_{i=1}^{16} \alpha_i) \mod \xi$, and $\boldsymbol{\beta} = \phi(\sum_{i=1}^{16} \beta_i) \mod \xi$, respectively. Correctness directly follows by Theorem 4.1 (ϕ, Tr are "commutative"), namely,

$$\phi(\mathsf{Tr}(x_0)) * \phi(\mathsf{Tr}(x_1)) = \phi(\sum_{i=1}^{16} \alpha_i) + \phi(\sum_{i=1}^{16} \beta_i) = \boldsymbol{\alpha} + \boldsymbol{\beta},$$

as $\phi(\mathsf{Tr}(x_0)), \phi(\mathsf{Tr}(x_1)) \in \mathbb{F}_2^N$, and the coefficients of ξ of $\phi(\sum_{i=1}^{16} \alpha_i), \phi(\sum_{i=1}^{16} \beta_i)$ vanish. For security, by the pseudorandomness of x_0, x_1, $\mathsf{Tr}(x_0), \mathsf{Tr}(x_1)$ are pseudorandom, and by the security of underlying SPFSS, $\boldsymbol{\alpha}, \boldsymbol{\beta}$ are indistinguishable from the uniform distribution over $\mathbb{F}_2^N$.

At a first glance, the seed size would be quite large, since term e.g., $s_0^2 s_1^2$ is the product of four t-sparse elements of $\mathcal{R}_2$ and could be t^4-sparse. In fact, we show a much tighter upper bound of the sparsity. That is quadratic of t. Due to Lemma 4.3, s_σ^2, e_σ^2 are also t-sparse in $\mathcal{R}_2$. Hence the seed size of our PCG is proportional to t^2, same as that of [BCCD23].

Optimizations. We first show an optimization that reduces the number of underlying FSS instances of point functions by half, from carefully rewriting Eq.(2). Jumping ahead, the goal is to distribute $\alpha, \beta \in \mathcal{R}_2$ to $\mathrm{P}_0, \mathrm{P}_1$, with the following:

$$\mathsf{Tr}(x_0) \cdot \mathsf{Tr}(x_1) = \mathsf{Tr}(\alpha) + \mathsf{Tr}(\beta). \tag{3}$$

Note that $\mathsf{Tr}(\alpha), \mathsf{Tr}(\beta)$ can be locally computed by $\mathrm{P}_0, \mathrm{P}_1$ respectively. To this end, by Definition 4.1, we can rewrite Eq.(2) as follows:

$$\begin{aligned}
&\mathsf{Tr}(x_0) \cdot \mathsf{Tr}(x_1) = (a^4 s_0^2 s_1^2 + a^2 s_0 s_1) + (a^3 s_0^2 s_1 + a^3 s_0 s_1^2) + (a^2 s_0^2 e_1^2 + a s_0 e_1) \\
&+ (a^2 s_0^2 e_1 + a s_0 e_1^2) + (a s_1 e_0 + a^2 s_1^2 e_0^2) + (a s_1 e_0^2 + a^2 s_1^2 e_0) + (e_0^2 e_1^2 + e_0 e_1) + (e_1^2 e_2 + e_1 e_2^2) \\
&= \mathsf{Tr}(a^2 s_0 s_1) + \mathsf{Tr}(s_0 s_1^2) + \mathsf{Tr}(a s_0 e_1) + \mathsf{Tr}(a s_0 e_1^2) + \mathsf{Tr}(a s_1 e_0) \\
&\quad + \mathsf{Tr}(a s_1 e_0^2) + \mathsf{Tr}(e_0 e_1) + \mathsf{Tr}(e_0 e_1^2).
\end{aligned} \tag{4}$$

Therefore, it suffices to additively share these eight t^2-sparse terms $s_0 s_1, s_0 s_1^2, s_0 e_1, s_0 e_1^2, s_1 e_0, s_1 e_0^2, e_0 e_1, e_1 e_2^2$ via SPFSS. We denote the shares held by $\mathrm{P}_0, \mathrm{P}_1$ as

α_i, β_i, $i \in [1, 8]$. Then party P_0 can locally compute

$$\alpha := a^2 \cdot \alpha_1 + a \cdot (\alpha_3 + \alpha_4 + \alpha_5 + \alpha_6) + (\alpha_2 + \alpha_7 + \alpha_8),$$

while P_1 locally computes

$$\beta := a^2 \cdot \beta_1 + a \cdot (\beta_3 + \beta_4 + \beta_5 + \beta_6) + (\beta_2 + \beta_7 + \beta_8).$$

For correctness, it is easy to verify that α, β satisfy Eq.(3). Then $(\mathsf{Tr}(x_0), \mathsf{Tr}(x_1), \mathsf{Tr}(\alpha), \mathsf{Tr}(\beta))$ are essentially N OLEs over $\mathbb{F}_2$. For security, α, β are indistinguishable from being uniformly random in $\mathcal{R}_2$ (as e.g., α_7, β_7 are uniformly random) by the security of underlying SPFSS. This completes our first optimization.

Next, we show that the above $\alpha, \beta \in \mathcal{R}_2$ essentially admit $2N$ independent random OLE correlations over $\mathbb{F}_2$, thus further reducing the communication by half in an amortization flavor. This optimization also relies on the properties of our trace function. Let $\xi \in \mathbb{F}_4$ s.t. $\mathbb{F}_4 = \mathbb{F}_2(\xi)$, then $(1, \xi)$ is a basis of $\mathbb{F}_4$ over $\mathbb{F}_2$. By Theorem 4.3, for $x \xleftarrow{\$} \mathcal{R}_2$, $\mathsf{Tr}(x)$ and $\mathsf{Tr}(\xi x)$ are independent and uniform over $\mathbb{F}_2^N$[3]. Hence, we can extract additional N OLEs from the following observation:

$$\begin{aligned}
&\mathsf{Tr}(\xi x_0) \cdot \mathsf{Tr}(\xi x_1) = (\xi^2 x_0^2 + \xi x_0)(\xi^2 x_1^2 + \xi x_1) \\
&= \prod_{\sigma \in \{0,1\}} (a^2 \cdot \xi^2 s_\sigma^2 + a \cdot \xi s_\sigma + \xi^2 e_\sigma^2 + \xi e_\sigma) \\
&= \mathsf{Tr}(a^2 \xi^2 \underbrace{s_0 s_1}_{\alpha_1+\beta_1}) + \mathsf{Tr}(\underbrace{s_0 s_1^2}_{\alpha_2+\beta_2}) + \mathsf{Tr}(a\xi^2 \underbrace{s_0 e_1}_{\alpha_3+\beta_3}) + \mathsf{Tr}(a \underbrace{s_0 e_1^2}_{\alpha_4+\beta_4}) \\
&\quad + \mathsf{Tr}(a\xi^2 \underbrace{s_1 e_0}_{\alpha_5+\beta_5}) + \mathsf{Tr}(a \underbrace{s_1 e_0^2}_{\alpha_6+\beta_6}) + \mathsf{Tr}(\xi^2 \underbrace{e_0 e_1}_{\alpha_7+\beta_7}) + \mathsf{Tr}(\underbrace{e_0 e_1^2}_{\alpha_8+\beta_8})
\end{aligned} \tag{5}$$

Therefore, the above $\alpha_i, \beta_i \in \mathcal{R}_2$ from Eq.(3) can be reused. Let P_0 additionally compute

$$\alpha' := a^2 \xi^2 \alpha_1 + a(\xi^2 \alpha_3 + \alpha_4 + \xi^2 \alpha_5 + \alpha_6) + (\alpha_2 + \xi^2 \alpha_7 + \alpha_8)$$

and P_1 additionally compute

$$\beta' := a^2 \xi^2 \beta_1 + a(\xi^2 \beta_3 + \beta_4 + \xi^2 \beta_5 + \beta_6) + (\beta_2 + \xi^2 \beta_7 + \beta_8).$$

It can be directly verified that

$$\mathsf{Tr}(\xi x_0) \cdot \mathsf{Tr}(\xi x_1) = \mathsf{Tr}(\alpha') + \mathsf{Tr}(\beta').$$

This implies extra N OLE correlations over $\mathbb{F}_2$ by the isomorphism. Security follows the security of underlying SPFSS and Theorem 4.3.

Intuitively, $\mathbb{F}_4$ can be viewed as a dimension 2 vector space over $\mathbb{F}_2$. From the isomorphism $\mathcal{R}_2 \simeq \mathbb{F}_4^N$, a random element x of $\mathcal{R}_2$ has $2N$-bit entropy and there

[3] Actually, this is nothing but the coordinates of x in the dual basis (with respect to the trace map) of $(1, \xi)$.

are at most $2N$ elements of $\mathbb{F}_2$ can be extracted from x. Analogously, $\mathcal{R}_k$ allows at most extractions of kN $\mathbb{F}_2$ elements.

Sharing products of sparse elements. We show how to share the cross-terms using a sum of point function secret sharing scheme SPFSS. Note that SPFSS can be directly instantiated with DPFs. For $\mathcal{R}_2 = \mathbb{F}_4[\mathtt{X}_1, \ldots, \mathtt{X}_n]/(\mathtt{X}_1^3 - 1, \ldots, \mathtt{X}_n^3 - 1)$, it is easy to see that $\mathcal{R}_2$ has a basis $\{\prod_{i=1}^n \mathtt{X}_i^{j_i}\}$, where $(j_1, \ldots, j_n)$ goes through $[0, 3)^n$. For simplicity, we define a bijection $\varphi_2 : j \mapsto \prod_{i=1}^n \mathtt{X}_i^{j_i}$, where $j \in [0, N)$ and $N = 3^n$. Given the bijection φ_2 and the basis $\{\prod_{i=1}^n \mathtt{X}_i^{j_i}\}$, any t-sparse element x of $\mathcal{R}_2$ can be uniquely represented by two vectors $\mathbf{A} \in [0, N)^t$ (indicating the non-zero positions of x) and $\boldsymbol{s} \in (\mathbb{F}_4^*)^t$ (corresponding to the coefficients of x). This allows to succinctly distribute additive shares of $x \in \mathcal{R}_2$ via SPFSS with domain $[0, N)$ and range $\mathbb{F}_4$.

Let $(\mathbf{A}_\sigma^0, \boldsymbol{s}_\sigma)$ represent the t-sparse element s_σ of $\mathcal{R}_2$, for $\sigma \in \{0, 1\}$. By Lemma 4.3, s_1^2 is t-sparse of $\mathcal{R}_2$ as well. Furthermore, the representation of s_1^2, denoted by $((\mathbf{A}_1^0)^2, (\boldsymbol{s}_1)^2)$, can be computed as

$$(\mathbf{A}_1^0)^2 := \varphi_2^{-1}\big(\varphi_2(\mathbf{A}_1^0) * \varphi_2(\mathbf{A}_1^0)\big), \quad (\boldsymbol{s}_1)^2 = \boldsymbol{s}_1 * \boldsymbol{s}_1.$$

Then, by Lemma 4.3, $s_0 s_1^2$ is t^2-sparse of $\mathcal{R}_2$, and the representation of $s_0 s_1^2$, denoted by $(\mathbf{A}_0^0 \boxplus (\mathbf{A}_1^0)^2, \boldsymbol{s}_0 \otimes (\boldsymbol{s}_1)^2)$, can be computed as follows:

$$\mathbf{A}_0^0 \boxplus (\mathbf{A}_1^0)^2 := \varphi_2^{-1}\big(\varphi_2(\mathbf{A}_0^0) \otimes \varphi_2((\mathbf{A}_1^0)^2)\big),$$

where $\otimes$ refers to the tensor product. Essentially, the above "additions" on the positions of $[0, N)$ are equivalent to the additions on $\mathbb{G} = \mathbb{Z}_3^n$, and we abuse the equivalence in this work. We remark that there exists a ternary FSS with domain $\mathbb{Z}_3^n$ [BBC+24, Section 5.1].

The final optimized construction. Though, we already get desired OLE correlations over $\mathbb{F}_2$ from $k = 2$, it is not easy to extend to a general $k \geq 2$. It seems very heuristic to write a similar formula for $k > 2$ as Eq.(4).

One significantly useful property of trace over finite fields is that for an arbitrary $u \in \mathbb{F}_{2^k}$, $\mathsf{Tr}(u) \in \mathbb{F}_2$, hence $\forall u, v \in \mathbb{F}_{2^k}$, $\mathsf{Tr}(u) \cdot \mathsf{Tr}(v) = \mathsf{Tr}(\mathsf{Tr}(u) \cdot v)$. To further exploit the properties of trace functions over $\mathcal{R}_k$, we establish the following result: $\forall f, g \in \mathcal{R}_k$, it holds that

$$\mathsf{Tr}(f) \cdot \mathsf{Tr}(g) = \mathsf{Tr}(\mathsf{Tr}(f) \cdot g) = \mathsf{Tr}(f \cdot \mathsf{Tr}(g)),$$

i.e., $\mathsf{Tr}(\cdot)$ over $\mathcal{R}$ "absorbs" $\mathsf{Tr}(\mathcal{R})$ as well. Moreover, our $\mathsf{Tr}(\cdot)$ over $\mathcal{R}_k$ also enjoys additive homomorphism, making it compatible with the additive sharings. Specifically, if x is additively shared over $\mathcal{R}_k$, then additive shares of $\mathsf{Tr}(x)$ can be locally obtained. These indicate that the trace function over $\mathcal{R}_k$ employs analogues properties of the trace over finite fields. Generally, $\mathsf{Tr}(\cdot)$ over $\mathcal{R}_k$ is properly defined with respect to the isomorphism ϕ.

Recall that it suffices to share $\mathsf{Tr}(x_0) \cdot \mathsf{Tr}(x_1)$, which offers kN extractions of OLEs over $\mathbb{F}_2$ by fixing a basis $(1, \xi, \ldots, \xi^{k-1})$ of $\mathbb{F}_{2^k} = \mathbb{F}_2(\xi)$. For an arbitrary

$k \geq 2$, and $j = 0, 1, \ldots, k-1$, we give the following concise representation:

$$\begin{aligned}
&\mathsf{Tr}(\xi^j x_0) \cdot \mathsf{Tr}(\xi^j x_1) = \mathsf{Tr}(\xi^j x_0 \cdot \mathsf{Tr}(\xi^j x_1)) \\
&\quad = \mathsf{Tr}\big(\xi^j \cdot (as_0 + e_0) \cdot \sum_{i=0}^{k-1} \xi^{j \cdot 2^i} \cdot (as_1 + e_1)^{2^i}\big) \\
&\quad = \mathsf{Tr}\big(\xi^{j \cdot (2^i+1)} \cdot \sum_{i=0}^{k-1} (as_0 + e_0)(a^{2^i} s_1^{2^i} + e_1^{2^i})\big) \\
&\quad = \sum_{i=0}^{k-1} \mathsf{Tr}\big(\xi^{j \cdot (2^i+1)} \cdot (a^{2^i+1} \cdot s_0 s_1^{2^i} + a^{2^i} \cdot e_0 s_1^{2^i} + a \cdot s_0 e_1^{2^i} + e_0 e_1^{2^i})\big).
\end{aligned}$$

The construction also satisfies programmability, allowing for generating multi-party Beaver triples over $\mathbb{F}_2$ as shown in [BCG+19b]. Compared to the approach of [BBC+24], the generation of multi-party $\mathbb{F}_2$ Beaver triples is *silent*, meaning that as long as the PCG seeds are honestly distributed, the parties can obtain Beaver triples via local expansion. We defer arguments of programmability to Section 5.

2.4 PCG for Authenticated Multiplication Triples.

Adapting the above OLE constructions to triples is straightforward, in which one can distribute additive sharings of x_0, x_1 via SPFSS. Sharing with a global key $\Delta \xleftarrow{\$} \mathbb{F}_{p^k}$ is much more complicated, where $k = O(\lambda)$. The above optimization of moving $\mathsf{Tr}(x_1)$ inside $\mathsf{Tr}(\cdot)$ does not work in this case as $\Delta \in \mathbb{F}_{p^k}$. Hence, the seed size grows by a factor of $O(k)$ in general. For $\sigma \in \{0, 1\}$ and $s \in [0, k-1]$, sharings of $\Delta \cdot \mathsf{Tr}(\xi^s \cdot x_\sigma)$ can be obtained as follows:

$$\begin{aligned}
\Delta \cdot \mathsf{Tr}(\xi^s \cdot x_\sigma) &= \sum_{i \in [0,k-1]} \Delta \cdot \xi^{s \cdot p^i} \cdot (as_\sigma + e_\sigma)^{p^i} \\
&= \sum_{i \in [0,k-1]} \xi^{s \cdot p^i} \cdot (a^{p^i} \Delta \cdot s_\sigma^{p^i} + \Delta \cdot e_\sigma^{p^i}).
\end{aligned}$$

Hence, it suffices to share $\Delta \cdot s_\sigma^{p^i}, \Delta \cdot e_\sigma^{p^i}$ via SPFSS, for all $i \in [0, k-1]$, which are t-sparse elements. As for sharings of $\Delta \cdot \mathsf{Tr}(\xi^s \cdot x_0) \cdot \mathsf{Tr}(\xi^s \cdot x_1)$, $s \in [0, k-1]$,

$$\begin{aligned}
&\Delta \cdot \mathsf{Tr}(\xi^s \cdot x_0) \cdot \mathsf{Tr}(\xi^s \cdot x_1) \\
&= \sum_{i,j \in [0,k-1]} \Delta \cdot \xi^{s \cdot p^i} \cdot (as_0 + e_0)^{p^i} \cdot \xi^{s \cdot p^j} \cdot (as_1 + e_1)^{p^j} \\
&= \sum_{i,j \in [0,k-1]} \xi^{s(p^i+p^j)} \cdot (a^{p^i+p^j} \Delta s_0^{p^i} s_1^{p^j} + a^{p^j} \Delta e_0^{p^i} s_1^{p^j} + a^{p^i} \Delta s_0^{p^i} e_1^{p^j} + \Delta e_0^{p^i} e_1^{p^j})
\end{aligned}$$

Hence, it suffices to share $\Delta \cdot s_0^{p^i} s_1^{p^j}, \Delta \cdot s_0^{p^i} e_1^{p^j}, \Delta \cdot e_0^{p^i} s_1^{p^j}, \Delta \cdot e_0^{p^i} e_1^{p^j}$ via SPFSS, for all $i, j \in [0, k-1]$, which are t^2-sparse elements. Putting all pieces together, the seed

size is $O\big((t^2k^2+t^2k+tk+t)\lambda \log N\big) = O(\lambda^3 k^2 \log N)$ for kN authenticated triples. In addition, we remark that the above construction is only for illustrating the basic idea and k does not have to be $O(\lambda)$. In fact, through our optimizations, k can be arbitrary integers larger than 2, so that the seed size remains $O(\lambda^3 k^2 \log N)$.

We also demonstrate an effcient maliciously secure setup protocol to produce the PCG seeds, via instantiating SPFSS with concretely efficient DPFs.

3 Preliminaries

Notations. Let $\mathbf{A}[i]$ denote the i-th entry of $\mathbf{A}$. Given $\ell, r \in \mathbb{Z}$, denote $[\ell, r] = [\ell, r+1) \subset \mathbb{Z}$ as the set of integers starts from ℓ and ends with r. If a value x is additively shared, we denote it as $[x]$. If a value x is shared via a SPDZ-style authenticated sharing, for instance $[\![x]\!] := ([\![x]\!]_0, [\![x]\!]_1)$ and $[\![x]\!]_\sigma := (x_\sigma, M_\sigma[x])$ where $x = x_0 + x_1$, $M_0[x] + M_1[x] = x \cdot (\Delta_0 + \Delta_1)$ and (Δ_0, Δ_1) is a sharing of the global key.

3.1 Function Secret Sharing

Function secret sharing (FSS) [BGI15, BGI16] allows to share functions succinctly among two parties. More concisely, for some secret function $f : I \to \mathbb{G}$ where $\mathbb{G}$ is an Abelian group, an FSS scheme splits it into two functions f_0, f_1, such that: (1) $f_0(x) + f_1(x) = f(x)$ for every input $x \in I$, and (2) f_0, f_1 can be represented by short keys K_0, K_1 respectively, with each of K_0, K_1 individually hiding f.

Distributed point function (DPF) [GI14, BGI15] is a useful and fundamental kind of FSS schemes, specialized for sharing point functions. We give a formal definition of DPF as follows.

Definition 3.1 (Distributed point function). *Given an input domain $I = [0, N)$, and an Abelian group $(\mathbb{G}, +)$, the point function $f_{\alpha,\beta} : I \to \mathbb{G}$ is defined by $f_{\alpha,\beta}(x) = \beta$ if $x = \alpha$, and $f_{\alpha,\beta}(x) = 0$ if $x \neq \alpha$, where $\alpha \in I, \beta \in \mathbb{G}$. A DPF scheme for the class of point functions $\{f_{\alpha,\beta} : \alpha \in I, \beta \in \mathbb{G}\}$ consists of the following two algorithms.*

- $\mathsf{DPF.Gen}(1^\lambda, \alpha, \beta)$ *is a PPT algorithm that, given the security parameter λ, a position index $\alpha \in I$ and an element $\beta \in \mathbb{G}$, outputs a pair of keys $(\mathbf{k}_0^{\mathrm{dpf}}, \mathbf{k}_1^{\mathrm{dpf}})$.*
- $\mathsf{DPF.Eval}(\sigma, \mathbf{k}_\sigma^{\mathrm{dpf}}, x)$ *is a polynomial-time algorithm that, given a key $\mathbf{k}_\sigma^{\mathrm{dpf}}$ for party $\sigma \in \{0, 1\}$, and an input $x \in I$, outputs $z_\sigma \in \mathbb{G}$.*

The above scheme should satisfy the following requirements.

- **Correctness:** For any $\alpha \in I, \beta, x \in \mathbb{G}$, it holds that $z_0 + z_1 = f(x)$, where $z_\sigma \leftarrow \mathsf{DPF.Eval}(\sigma, \mathbf{k}_\sigma^{\mathrm{dpf}}, x)$ for $\sigma \in \{0, 1\}$ and $(\mathbf{k}_0^{\mathrm{dpf}}, \mathbf{k}_1^{\mathrm{dpf}}) \xleftarrow{\$} \mathsf{DPF.Gen}(1^\lambda, \alpha, \beta)$.
- **Security:** For any $\sigma \in \{0, 1\}$, there exists a PPT simulator $\mathcal{S}$ such that for any point function $f_{\alpha,\beta}$, the distributions $\{(\mathbf{k}_\sigma^{\mathrm{dpf}} : (\mathbf{k}_0^{\mathrm{dpf}}, \mathbf{k}_1^{\mathrm{dpf}}) \xleftarrow{\$} \mathsf{DPF.Gen}(1^\lambda, \alpha, \beta))\}$ and $\{\mathbf{k}_\sigma^{\mathrm{dpf}} \xleftarrow{\$} \mathcal{S}(1^\lambda, I, \mathbb{G})\}$ are computationally indistinguishable.

In this paper, we will use a simple and generic extension of DPF to sums of point functions. Let $\mathbf{A} = (\alpha_1, \ldots, \alpha_t) \in [0, N)^t$, $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_t) \in \mathbb{G}^t$, the sum of point functions $f_{\mathbf{A},\boldsymbol{\beta}} : [0, N) \to \mathbb{G}$ is defined by $\sum_{i=1}^{t} f_{\alpha_i,\beta_i}$. And it is direct to see that an FSS for $f_{\mathbf{A},\boldsymbol{\beta}}$ can be realized by invoking t parallel DPFs, with each for f_{α_i,β_i}. Similarly, we define $(\mathsf{SPFSS.Gen}, \mathsf{SPFSS.Eval})$ as FSS for sums of point functions, with security following that of DPF. For convenience, we define SPFSS.FullEval as evaluations of SPFSS.Eval on all possible $x \in [0, N)$.

The typical DPF construction [BGI16] uses an arbitrary pseudorandom generator(PRG) $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda+2}$ and the parameter is given as follows. The key size is at most $\lceil \log N \rceil (\lambda+2)+\lambda+\lceil \log |\mathbb{G}| \rceil$ bits. Taking $m = \left\lceil \frac{\log |\mathbb{G}|}{\lambda+2} \right\rceil$, the key generation SPFSS.Gen algorithm makes at most $2(\lceil \log N \rceil + m)$ PRG calls and the full domain evaluation algorithm SPFSS.FullEval makes at most $N(1+m)$ PRG calls. For regular multi-point distributions, the number of PRG calls sheaves by a factor t. Employing batching codes, the SPFSS.FullEval algorithm can be asympcotically improved at the cost of more complicated seed generation algorithm. Recently, the SPFSS.Gen and SPFSS.FullEval algorithms are improved via a half-tree technique [GYW+23]. To show that our OLE and authenticated multiplication triples constructions are comparable to the previous constructions from Ring-LPN [BCG+20] and QA-SD [BCCD23], we still use the parameters from [BGI16] because the previous constructions rely on the parameters of [BGI16].

3.2 Pseudorandom Correlation Generators

The notion pseudorandom correlation generators PCG was first introduced in [BCG+19a, BCG+19b]. Informally, a PCG for some correlation, allows to distribute short, correlated seeds to the parties, such that each party can locally compute long, correlated pseudorandomness (satisfying the desired correlation) from expanding their own seed. To give a formal definition, we start with correlation generators, and reverse-sampleable correlation generators.

Definition 3.2 (Correlation generator). *A correlation generator $\mathcal{C}$ is a PPT algorithm that on input 1^λ, outputs a pair of strings in $\{0,1\}^\ell \times \{0,1\}^\ell$, where $\ell \in \mathsf{poly}(\lambda)$.*

Definition 3.3 (Reverse-sampleable correlation generator). *Let $\mathcal{C}$ be a correlation generator. We say $\mathcal{C}$ is reverse sampleable if there exists a* PPT *algorithm* RSample *such that for $\sigma \in \{0,1\}$, the following distribution*

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

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

In the following, we give the definition of PCG. Programmable PCG is formally defined in Appendix A.

Definition 3.4 (PCG). *Let $\mathcal{C}$ be a reverse-sampleable correlation generator. A PCG for $\mathcal{C}$ consists of two algorithms* $(\mathsf{PCG.Gen}, \mathsf{PCG.Expand})$ *with the following syntax:*

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

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

- **Correctness.** The following distribution

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

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

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

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

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

3.3 Syndrome Decoding of Quasi-Abelian Codes

In this section, we recall the syndrome decoding assumption of Quasi-Abelian codes introduced in [BCCD23]. For simplicity, we only introduce the concrete instantiation used in this work, and refer details of Quasi-Abelian codes to [BCCD23, BBC+24]. We define Ring-LPN assumption in a similar way (viewed as a univariate variant of QA-SD) in Appendix A.

Definition 3.5 (Search QA-SD). *Let $\mathcal{R} = \mathbb{F}_q[\mathtt{X}_1, \ldots, \mathtt{X}_n]/(\mathtt{X}_1^d - 1, \ldots, \mathtt{X}_n^d - 1)$, where $d \mid q-1$. Let $c \geq 2$ be some constant integer called the compression factor. Let $\boldsymbol{a} = (1, a_1, \ldots, a_{c-1})$, where $a_i \xleftarrow{\$} \mathcal{R}$, $i \in [1, c-1]$. Let $e_0, e_1, \ldots, e_{c-1}$ be random t-sparse elements of $\mathcal{R}$. Given access to a pair of the form $(\boldsymbol{a}, \langle \boldsymbol{a}, \boldsymbol{e} \rangle)$, the goal is to recover $\boldsymbol{e}$.*

Definition 3.6 (Decisional QA-SD). *Let $\mathcal{R} = \mathbb{F}_q[\mathtt{X}_1, \ldots, \mathtt{X}_n]/(\mathtt{X}_1^d - 1, \ldots, \mathtt{X}_n^d - 1)$, where $d \mid q-1$. Let $c \geq 2$ be some constant integer called the compression factor. The goal is to distinguish the following two distributions:*

$$\mathcal{D}_0 : \{(a_1, \ldots, a_{c-1}, u)\}, \text{ where } a_i, u \xleftarrow{\$} \mathcal{R}, i \in [1, c-1]$$

$$\mathcal{D}_1 : \{(a_1, \ldots, a_{c-1}, \langle \boldsymbol{a}, \boldsymbol{e} \rangle + e_0)\}, \text{ where } a_i \xleftarrow{\$} \mathcal{R}, e_0, e_i \text{ are random } t\text{-sparse elements of } \mathcal{R}, i \in [1, c-1].$$

In this work, for simplicity we set the compression factor $c = 2$, and the above problems are referred as search QA-SD$(\mathcal{R}, t)$, decisional QA-SD$(\mathcal{R}, t)$, respectively. All of our constructions can be naturally extended to cases of $c \geq 2$. We say that the search (decisional) QA-SD$(\mathcal{R}, t)$ assumption holds when there exists no PPT algorithm that solves the problem with a non-negligible advantage.

4 Ring Isomorphisms and Trace Functions

In this section, we show some useful properties of the univariate ring and multivariate ring that we are working on. Then we define the trace functions over the ring, which employ similar properties to the trace function over finite field extension. The detailed proofs of the results in this section are presented in Appendix B. We first show the univariate ring and the multivariate ring are isomorphic to a vector ring.

Lemma 4.1 (The Univariate Ring). *Let $\mathcal{R} := \mathbb{F}_q[\mathtt{X}]/(\mathtt{X}^N - 1)$ with $N \mid (q-1)$. Then there exists $\phi : \mathcal{R} \to \mathbb{F}_q^N$ such that ϕ is a ring isomorphism. In particular, the addition and multiplication operations on $\mathbb{F}_q^N$ are defined component-wisely.*

Lemma 4.2 (The Multivariate Ring). *Let $\mathcal{R} = \mathbb{F}_q[\mathtt{X}_1, \ldots, \mathtt{X}_n]/(\mathtt{X}_1^d - 1, \ldots, \mathtt{X}_n^d - 1)$, where $d \mid (q-1)$. Then there exists $\phi : \mathcal{R} \to \mathbb{F}_q^N$, where $N = d^n$, such that ϕ is a ring isomorphism.*

Remark 4.1. Lemma 4.2 can be generalized to $\mathcal{R} = \mathbb{F}_q[\mathtt{X}_1 \ldots \mathtt{X}_t]/(\mathtt{X}_1^{d_1} - 1 \ldots \mathtt{X}_n^{d_n} - 1)$ with each $d_i \mid (q-1)$. Lemma 4.1 can be viewed as a special case of Lemma 4.2 with $n = 1$.

Let $\mathbb{F}_p$ be a prime field of characteristic p, and $\mathbb{F}_{p^k}$ be a finite extension of $\mathbb{F}_p$. Then we show the polynomial ring $\mathcal{R}$ over $\mathbb{F}_{p^k}$ inherits some nice properties from the finite field extension $\mathbb{F}_{p^k}$ of $\mathbb{F}_p$.

Lemma 4.3 (Properties of $\mathcal{R}$). *For a prime p and $k \in \mathbb{N}$, let $\mathcal{R} = \mathbb{F}_{p^k}[\mathtt{X}]/(\mathtt{X}^N - 1)$ or $\mathcal{R} = \mathbb{F}_{p^k}[\mathtt{X}_1, \ldots, \mathtt{X}_n]/(\mathtt{X}_1^d - 1, \ldots, \mathtt{X}_n^d - 1)$ as defined above.*

1. *$\mathcal{R}$ has characteristic p.*
2. *For arbitrary $a, b \in \mathcal{R}$ and $j \in \mathbb{N}$, then $(a+b)^{p^j} = a^{p^j} + b^{p^j}$.*
3. *If $a \in \mathcal{R}$ is t-sparse, then a^{p^j} is t-sparse for any integer j.*
4. *For arbitrary $a \in \mathcal{R}$, $a^{p^k} = a$.*

Recall that for arbitrary $\alpha \in \mathbb{F}_{p^k}$, the trace $\mathsf{Tr}_{\mathbb{F}_{p^k}/\mathbb{F}_p}(\alpha)$ of $\alpha \in \mathbb{F}_{p^k}$ over $\mathbb{F}_p$ is defined by

$$\mathsf{Tr}_{\mathbb{F}_{p^k}/\mathbb{F}_p}(\alpha) = \alpha + \alpha^p + \cdots + \alpha^{p^{k-1}}.$$

In this work, we focus on a general prime field $\mathbb{F}_p$ and the specific case where $p = 2$, and we simplify the notation $\mathsf{Tr}(\alpha)$ as the trace of $\alpha \in \mathbb{F}_{p^k}$ over $\mathbb{F}_p$. In addition, we naturally define the trace $\mathsf{Tr}(\alpha)$ of an vector $\alpha \in \mathbb{F}_{p^k}^N$ as $(\mathsf{Tr}(\alpha_1), \ldots, \mathsf{Tr}(\alpha_N))$.

Below we define trace functions over $\mathcal{R}$.

Definition 4.1 (Trace Functions for Ring Elements). *For an arbitrary $f \in \mathcal{R}$, define the trace function as $\mathsf{Tr}(f) := \sum_{i=0}^{k-1} f^{p^i}$.*

One significant property of trace functions over finite fields is that for an arbitrary $\alpha \in \mathbb{F}_{p^k}$, $\mathsf{Tr}(\alpha) \in \mathbb{F}_p$. We prove that our trace function over $\mathcal{R}$ is properly defined with respect to the isomorphism ϕ, i.e., $\forall f \in \mathcal{R}$, $\phi(\mathsf{Tr}(f)) \in \mathbb{F}_p^N$.

Theorem 4.1. *For an arbitrary $f \in \mathcal{R}$, $\phi(\mathsf{Tr}(f)) \in \mathbb{F}_p^N$. In particular,*

$$\phi(\mathsf{Tr}(f)) = \mathsf{Tr}(\phi(f)) \in \mathbb{F}_p^N.$$

Hence, one can view as ϕ and Tr are "commutative".

Moreover, the trace function for $\mathcal{R}$ also satisfies the following properties as the trace function for $\mathbb{F}_{p^k}$.

Theorem 4.2. *The trace function over $\mathcal{R}$ satisfies the following properties:*

1. $\forall f_1, f_2 \in \mathcal{R}$, $\mathsf{Tr}(f_1 + f_2) = \mathsf{Tr}(f_1) + \mathsf{Tr}(f_2)$.
2. $\forall \alpha \in \mathbb{F}_p, \forall f \in \mathcal{R}$, $\mathsf{Tr}(\alpha \cdot f) = \alpha \cdot \mathsf{Tr}(f)$.
3. $\forall f \in \mathcal{R}$, $\mathsf{Tr}(f^p) = \mathsf{Tr}(f)$.
4. $\forall f_1, f_2 \in \mathcal{R}$, $\mathsf{Tr}(f_1) \cdot \mathsf{Tr}(f_2) = \mathsf{Tr}(\mathsf{Tr}(f_1) \cdot f_2) = \mathsf{Tr}(f_1 \cdot \mathsf{Tr}(f_2))$.

Note that for arbitrary two field elements $\alpha, \beta \in \mathbb{F}_{p^k}$, $\mathsf{Tr}(\alpha) \cdot \mathsf{Tr}(\beta) = \mathsf{Tr}(\mathsf{Tr}(\alpha) \cdot \beta)$ because of $\mathsf{Tr}(\alpha) \in \mathbb{F}_p$. Generalizing it to vectors, for arbitrary two vectors $\mathbf{u}, \mathbf{v} \in \mathbb{F}_{p^k}^N$, it holds that $\mathsf{Tr}(\mathbf{u}) * \mathsf{Tr}(\mathbf{v}) = \mathsf{Tr}(\mathsf{Tr}(\mathbf{u}) * \mathbf{v})$ as the multiplication is computed component-wise. Property 4 of Theorem 4.2 indicates that a similar result holds for $\mathcal{R}$ as well.

Theorem 4.3. *Let $\xi \in \mathbb{F}_{p^k}$ s.t. $\mathbb{F}_{p^k} = \mathbb{F}_p(\xi)$. For $x \xleftarrow{\$} \mathcal{R}$, we have that $\big(\mathsf{Tr}(x), \mathsf{Tr}(\xi x), \ldots, \mathsf{Tr}(\xi^{k-1} x)\big)$ are distributed uniformly at random in $\mathbb{F}_p^{kN}$, where $N = d^n$.*

Theorem 4.3 essentially holds for any basis $(b_1, \ldots, b_k)$ of $\mathbb{F}_{p^k}$ over $\mathbb{F}_p$. For instance, $\big(\mathsf{Tr}(\xi^{p^0} x), \mathsf{Tr}(\xi^{p^1} x), \ldots, \mathsf{Tr}(\xi^{p^{k-1}} x)\big)$ are distributed uniformly at random in $\mathbb{F}_p^{kN}$ as well, since $(\xi^{p^0}, \xi^{p^1}, \ldots, \xi^{p^{k-1}})$ is a basis.

Theorem 4.4. *Let k, η, d be integers such that $d \mid p^k - 1$ and $k \mid \eta$. Then the isomorphism $\phi_k : \mathcal{R}_k = \mathbb{F}_{p^k}[\mathtt{X}_1, \ldots, \mathtt{X}_n]/(\mathtt{X}_1^d - 1, \ldots, \mathtt{X}_n^d - 1) \to \mathbb{F}_{p^k}^N$ naturally induces an isomorphism $\phi_\eta : \mathcal{R}_\eta = \mathbb{F}_{p^\eta}[\mathtt{X}_1, \ldots, \mathtt{X}_n]/(\mathtt{X}_1^d - 1, \ldots, \mathtt{X}_n^d - 1) \to \mathbb{F}_{p^\eta}^N$, where $N = d^n$.*

5 PCG for OLE over Any Finite Field

In this section, we show how to construct a secure programmable PCG for OLE over any field from using *Trace*. To address difference from the state-of-the-art construction in [BCCD23] that only supports a field $\mathbb{F}$ with $|\mathbb{F}| > 2$, we discuss the particularly interesting case of OLE over $\mathbb{F}_2$ in Section 2.3, referred as a

warm-up construction. Our trace approach essentially can apply to any field, and the core idea is to use trace to locally and efficiently convert correlations over $\mathbb{F}_{p^k}$ to the target OLE correlations over $\mathbb{F}_p$.

In this section, we give a generic PCG construction for OLE over any field. Basically, the construction is obtained by a similar approach as in Section 2.3.

Let $\mathcal{R}_{p,k}^{n,d} = \mathbb{F}_{p^k}[\mathtt{X}_1, \ldots, \mathtt{X}_n]/(\mathtt{X}_1^d - 1, \ldots, \mathtt{X}_n^d - 1)$ with $\mathbb{F}_p$ an arbitrary finite field of characteristic p and $d \mid p^k - 1$. W.l.o.g., assume d reaches the upper bound $p^k - 1$ and simply write $\mathcal{R}_{p,k}^{n,d}$ by $\mathcal{R}_k$. Assume QA-SD$(\mathcal{R}_k, t)$, and we show how to generate kN OLEs over $\mathbb{F}_p$ in a batch, where $N = (p^k - 1)^n$. Let $a \xleftarrow{\$} \mathcal{R}_k$ be the public input, and s_σ, e_σ be random t-sparse elements of $\mathcal{R}_k$, where $\sigma \in \{0, 1\}$. By hardness of QA-SD$(\mathcal{R}_k, t)$, $x_\sigma := a \cdot s_\sigma + e_\sigma$ is pseudorandom for $\sigma \in \{0, 1\}$. Similar to our initial idea of Section 2.3, we have the following observation:

$$
\begin{aligned}
&\mathsf{Tr}(x_0) \cdot \mathsf{Tr}(x_1) = (x_0^{p^{k-1}} + \ldots + x_0^{p^0}) \cdot (x_1^{p^{k-1}} + \ldots + x_1^{p^0}) \\
&= (\sum_{i \in [0,k-1]} {x_0}^{p^i}) \cdot (\sum_{i \in [0,k-1]} {x_1}^{p^i}) = \sum_{i,j \in [0,k-1]} {x_0}^{p^i} {x_1}^{p^j} \\
&= \sum_{i,j \in [0,k-1]} (as_0 + e_0)^{p^i} \cdot (as_1 + e_1)^{p^j} \\
&= \sum_{i,j \in [0,k-1]} (a^{p^i} s_0^{p^i} + e_0^{p^i}) \cdot (a^{p^j} s_1^{p^j} + e_1^{p^j}) \\
&= \sum_{i,j \in [0,k-1]} (a^{p^i + p^j} s_0^{p^i} s_1^{p^j} + a^{p^j} e_0^{p^i} s_1^{p^j} + a^{p^i} s_0^{p^i} e_1^{p^j} + e_0^{p^i} e_1^{p^j})
\end{aligned}
\tag{6}
$$

Thus, one can obtain additive shares of $\mathsf{Tr}(x_0) \cdot \mathsf{Tr}(x_1)$ by succinctly sharing these $4k^2$ cross-terms via SPFSS. In fact, we can significantly reduce the cross terms needs to be shared by exploiting the $\mathbb{F}_p$-linearity of trace, which is similar to that of our first optimization in Section 2.3. The difference is that here we present in a more systematic way, by Theorem 4.2 (Property 4).

$$
\begin{aligned}
&\mathsf{Tr}(x_0) \cdot \mathsf{Tr}(x_1) = \mathsf{Tr}(x_0 \cdot \mathsf{Tr}(x_1)) = \mathsf{Tr}(x_0 \cdot (\sum_{i=0}^{k-1} x_1^{p^i})) = \sum_{i=0}^{k-1} \mathsf{Tr}(x_0 \cdot x_1^{p^i}) \\
&= \sum_{i=0}^{k-1} \mathsf{Tr}\big((as_0 + e_0) \cdot (as_1 + e_1)^{p^i}\big) = \sum_{i=0}^{k-1} \mathsf{Tr}\big((as_0 + e_0) \cdot (a^{p^i} s_1^{p^i} + e_1^{p^i})\big) \\
&= \sum_{i=0}^{k-1} \mathsf{Tr}(a^{p^i+1} s_0 s_1^{p^i} + a^{p^i} e_0 s_1^{p^i} + a s_0 e_1^{p^i} + e_0 e_1^{p^i})
\end{aligned}
\tag{7}
$$

By Lemma 4.3, $s_1^{p^i}, e_1^{p^i}$ are t-sparse elements of $\mathcal{R}_k$ as well. Therefore, for each $i \in [0, k-1]$, we can use SPFSS to succinctly share the t^2-sparse elements $s_0 s_1^{p^i}, e_0 s_1^{p^i}, s_0 e_1^{p^i}, e_0 e_1^{p^i}$ of $\mathcal{R}_k$. Hence, the communication complexity scales with $4k$. By Theorem 4.3, essentially we can extract in total kN OLE correlations over $\mathbb{F}_p$ from these $4k$ shares. This time, we consider the basis $\xi^{p^0}, \xi^{p^1}, \ldots, \xi^{p^{k-1}}$

of $\mathbb{F}_{p^k} = \mathbb{F}_p(\xi)$ over $\mathbb{F}_p$. Then, for each $j \in [0, k-1]$, we have that

$$\begin{aligned}\mathsf{Tr}(\xi^{p^j} x_0) \cdot \mathsf{Tr}(\xi^{p^j} x_1) &= \sum_{i=0}^{k-1} \mathsf{Tr}\big(\xi^{p^j} \cdot (as_0 + e_0) \cdot \xi^{p^{j+i}} \cdot (as_1 + e_1)^{p^i}\big) \\ &= \sum_{i=0}^{k-1} \mathsf{Tr}\big(\xi^{p^j + p^{j+i}} \cdot (a^{p^i+1} s_0 s_1^{p^i} + a^{p^i} e_0 s_1^{p^i} + a s_0 e_1^{p^i} + e_0 e_1^{p^i})\big).\end{aligned} \tag{8}$$

With Eq.(7) & (8) as above, we are able to explicitly present a construction $\mathsf{Cons}_{\mathsf{OLE}}^{\mathbb{F}_p}$ that works for any $p, k \geq 2$. The construction relies on the hardness of QA-SD$(\mathcal{R}_k, t)$ and an SPFSS scheme for sums of point functions, with domain $[0, N)$ and range $\mathbb{F}_{p^k}$. We address that here we define a bijection $\varphi_k : j \mapsto \prod_{i=1}^n \mathtt{X}_i^{j_i}$, where $j \in [0, N)$ and $(j_1, \ldots, j_n)$ go through $[0, p^k - 1)^n$. Thus, any $s \in \mathcal{R}_k$ corresponds to a unique pair $(\mathbf{A}, \boldsymbol{s}) \in [0, N)^t \times \mathbb{F}_{p^k}^t$ under φ_k. Similarly, for $\mathbf{A}_0, \mathbf{A}_1 \in [0, N)^t$, we define $\mathbf{A}_0 \boxplus \mathbf{A}_1 := \varphi_k^{-1}\big(\varphi_k(\mathbf{A}_0) \otimes \varphi_k(\mathbf{A}_1)\big)$, and the operation $(\mathbf{A}_0)^{p^i}$ for $i \in [0, k-1]$ is generalized from that of Section 2.3 in a similar way. Note that the above "additions" on the positions of $[0, N)$ are essentially nothing more than standard additions on a group $\mathbb{G} = \mathbb{Z}_{p^k-1}^n$.

Construction 1: $\mathsf{Cons}_{\mathsf{OLE}}^{\mathbb{F}_p}$

PARAMETER: Security parameter λ, noise weight $t = t(\lambda)$, $N = (p^k - 1)^n$, $\mathcal{R}_k = \mathbb{F}_{p^k}[\mathtt{X}_1, \ldots, \mathtt{X}_n]/(\mathtt{X}_1^{p^k-1} - 1, \ldots, \mathtt{X}_n^{p^k-1} - 1)$, $\xi \in \mathbb{F}_{p^k}$ s.t. $\mathbb{F}_{p^k} = \mathbb{F}_p(\xi)$. An FSS scheme $(\mathsf{SPFSS.Gen}, \mathsf{SPFSS.FullEval})$ for sums of t^2 point functions, with domain $[0, N)$[a] and range $\mathbb{F}_{p^k}$. Fix a basis $(\xi, \xi^{p^1}, \ldots, \xi^{p^{k-1}})$ of $\mathbb{F}_{p^k}$ over $\mathbb{F}_p$.
PUBLIC INPUT: A uniformly random $a \in \mathcal{R}_k$.
CORRELATION: After expansion, outputs $(\mathbf{x}_0^{(0)}, \ldots, \mathbf{x}_0^{(k-1)}, \mathbf{z}_0^{(0)}, \ldots, \mathbf{z}_0^{(k-1)}) \in \mathbb{F}_p^{2kN}$ and $(\mathbf{x}_1^{(0)}, \ldots, \mathbf{x}_1^{(k-1)}, \mathbf{z}_1^{(0)}, \ldots, \mathbf{z}_1^{(k-1)}) \in \mathbb{F}_p^{2kN}$ such that $\mathbf{x}_0^{(j)} \cdot \mathbf{x}_1^{(j)} = \mathbf{z}_0^{(j)} + \mathbf{z}_1^{(j)}$ for all $j \in [0, k)$.
Gen: On input 1^λ:

1. For $\sigma \in \{0, 1\}$, sample random vectors $\mathbf{A}_\sigma^0, \mathbf{A}_\sigma^1 \leftarrow [0, N)^t$ and $\boldsymbol{s}_\sigma, \boldsymbol{e}_\sigma \leftarrow (\mathbb{F}_{p^k}^*)^t$.
2. Sample FSS keys according to Eq.(7), namely for each $i \in [0, k-1]$:
   $(K_0^{4i}, K_1^{4i}) \xleftarrow{\$} \mathsf{SPFSS.Gen}(1^\lambda, \mathbf{A}_0^0 \boxplus (\mathbf{A}_1^0)^{p^i}, \boldsymbol{s}_0 \otimes (\boldsymbol{s}_1)^{p^i})$,
   $(K_0^{4i+1}, K_1^{4i+1}) \xleftarrow{\$} \mathsf{SPFSS.Gen}(1^\lambda, \mathbf{A}_0^1 \boxplus (\mathbf{A}_1^0)^{p^i}, \boldsymbol{e}_0 \otimes (\boldsymbol{s}_1)^{p^i})$,
   $(K_0^{4i+2}, K_1^{4i+2}) \xleftarrow{\$} \mathsf{SPFSS.Gen}(1^\lambda, \mathbf{A}_0^0 \boxplus (\mathbf{A}_1^1)^{p^i}, \boldsymbol{s}_0 \otimes (\boldsymbol{e}_1)^{p^i})$,
   $(K_0^{4i+3}, K_1^{4i+3}) \xleftarrow{\$} \mathsf{SPFSS.Gen}(1^\lambda, \mathbf{A}_0^1 \boxplus (\mathbf{A}_1^1)^{p^i}, \boldsymbol{e}_0 \otimes (\boldsymbol{e}_1)^{p^i})$,
3. For $\sigma \in \{0, 1\}$, let $\mathbf{k}_\sigma = \Big((K_\sigma^i)_{i \in [0, 4k-1]}, (\mathbf{A}_\sigma^0, \boldsymbol{s}_\sigma), (\mathbf{A}_\sigma^1, \boldsymbol{e}_\sigma)\Big)$.
4. Output $(\mathbf{k}_0, \mathbf{k}_1)$.

Expand: On input $(\sigma, \mathbf{k}_\sigma)$:

1. Parse $\mathbf{k}_\sigma$ as $\Big((K_\sigma^i)_{i \in [0, 4k-1]}, (\mathbf{A}_\sigma^0, \boldsymbol{s}_\sigma), (\mathbf{A}_\sigma^1, \boldsymbol{e}_\sigma)\Big)$.

2. Define elements of $\mathcal{R}_k$

$$s_\sigma = \sum_{l=1}^{t} \boldsymbol{s}_\sigma[l] \cdot \mathbf{A}_\sigma^0[l], \quad e_\sigma = \sum_{l=1}^{t} \boldsymbol{e}_\sigma[l] \cdot \mathbf{A}_\sigma^1[l].$$

3. For each $j \in [0, k-1]$, compute $\mathbf{x}_\sigma^{(j)} = \mathsf{Tr}\big(\xi^{p^j} \cdot (as_\sigma + e_\sigma)\big)$.
4. For $i \in [0, 4k-1]$, compute $u_{\sigma,i} \leftarrow \mathsf{SPFSS.FullEval}(\sigma, K_\sigma^i)$,viewed as $\mathcal{R}_k$ elements.
5. For each $j \in [0, k-1]$, according to Eq.(8), compute

$$\mathbf{z}_\sigma^{(j)} := \sum_{i=0}^{k-1} \mathsf{Tr}\big(\xi^{p^j+p^{j+i}} \cdot (a^{p^i+1}u_{\sigma,4i} + a^{p^i}u_{\sigma,4i+1} + au_{\sigma,4i+2} + u_{\sigma,4i+3})\big)$$

6. Output $\{(\mathbf{x}_\sigma^{(j)}, \mathbf{z}_\sigma^{(j)})\}_{j\in[0,k-1]}$.

[a] Each $j \in [0, N)$ corresponds to a basis $\prod_{i=1}^n \mathtt{x}_i^{j_i}$ of $\mathcal{R}_k$, where $j_i \in [0, p^k - 1)$. The addition on $[0, N)$ is actually the addition on $\mathbb{G} = \mathbb{Z}_{p^k-1}^n$.

Theorem 5.1. *Assume a secure* FSS *scheme* SPFSS *for sums of point functions and QA-SD$(\mathcal{R}_k, t)$ is hard. Then there exists a PCG construction* $\mathsf{Cons}_{\mathsf{OLE}}^{\mathbb{F}_p}$ *that generates OLE correlations over $\mathbb{F}_p$. If the* SPFSS *is based on a* $\mathsf{PRG} : \{0,1\}^\lambda \to \{0,1\}^{2\lambda+2}$ *via the* PRG*-based construction from [BGI16], for generating kN OLE correlations over $\mathbb{F}_p$, we have that:*

- *Each party's seed has maximum size around:* $4kt^2\big((\log N - \log t + 1)(\lambda+2) + \lambda + k\log p\big) + 2t(\log N + k\log p)$ *bits.*
- *The computation of* ***Expand*** *can be done with at most* $(2+\lceil(k\log p)/\lambda\rceil)4kNt$ PRG *operations and* $O(k^2 N \log N)$ *operations in* $\mathbb{F}_{p^k}$.

Remark 5.1. Regarding the size of the different parameters, we also follow optimizations of [BCG+20, BCCD23], e.g., assuming the QA-SD assumption holds also for a regular noise distributions. Then the seed size and computations can be obtained by following a similar argument as theirs.

Proof. Correctness is straightforward given by the Construction $\mathsf{Cons}_{\mathsf{OLE}}^{\mathbb{F}_p}$and Eq.(8). Security is based on a QA-SD assumption and the underlying SPFSS scheme, and we prove in a similar way to those of [BCG+20, BCCD23]. For completeness, we give a sketched proof as follows. It suffices to consider Party P_0's view (i.e., $\sigma = 0$). Let $(\mathbf{k}_0, \mathbf{k}_1) \xleftarrow{\$} \mathsf{PCG.Gen}(1^\lambda)$ with associated expanded outputs $(\mathbf{x}_0^{(j)}, \mathbf{z}_0^{(j)})$ and $(\mathbf{x}_1^{(j)}, \mathbf{z}_1^{(j)})$, for $j \in [0, k-1]$. We need to show that

$$\{(\mathbf{k}_0, (\mathbf{x}_1^{(j)}, \mathbf{z}_1^{(j)})_{j\in[0,k-1]})\} \equiv \left\{(\mathbf{k}_0, (\tilde{\mathbf{x}}_1^{(j)}, \tilde{\mathbf{z}}_1^{(j)})_{j\in[0,k-1]}) \mid \begin{array}{l} j \in [0,k-1], \tilde{\mathbf{x}}_1^{(j)} \xleftarrow{\$} \mathbb{F}_p^N, \\ \tilde{\mathbf{z}}_1^{(j)} = \mathbf{x}_0^{(j)} * \tilde{\mathbf{x}}_1^{(j)} - \mathbf{z}_0^{(j)} \end{array}\right\}$$

To show this, we use a sequence of hybrid distributions.

– Replace $\mathbf{z}_1^{(j)}$ by $\mathbf{x}_0^{(j)} * \mathbf{x}_1^{(j)} - \mathbf{z}_0^{(j)}$, for all $j \in [0, k-1]$.
– Step by step replace each the SPFSS key K_0^i in $\mathbf{k}_0$ by a simulated key generated only with the range and the domain of the function. Due to the correctness and the security properties of the SPFSS scheme, this distribution is indistinguishable from the original distribution.
– Replace $\mathbf{x}_1^{(j)}$ by a fresh $\tilde{\mathbf{x}}_1^{(j)}$ for all $j \in [0, k-1]$. It is also impossible to distinguish this distribution from the previous one, since the K_0^i are now completely independent of $(\mathbf{x}_1^{(0)}, \ldots, \mathbf{x}_1^{(k-1)})$, and we can rely on the QA-SD assumption and Theorem 4.3.
– Reverse step 2 by using the SPFSS security property once again.

We remark that QA-SD assumption implies $x_1 \in \mathcal{R}_k$ can not be distinguished from uniform, and Theorem 4.3 implies that $(\mathbf{x}_1^{(0)}, \ldots, \mathbf{x}_1^{(k-1)})$ computed from a random $x_1 \in \mathcal{R}_k$ are uniformly distributed in $\mathbb{F}_p^{kN}$. □

Note that to achieve security, choosing $t = O(\lambda)$ is sufficient. Actually $p = 2, k = 2$ is the case of our warm-up construction in Section 2.3. Thus analyses of $\mathsf{Cons}_{\mathsf{OLE}}^{\mathbb{F}_p}$ naturally apply to that case.

Theorem 5.2. *The PCG construction* $\mathsf{Cons}_{\mathsf{OLE}}^{\mathbb{F}_p}$ *has programmability.*

Proof. To show that $\mathsf{Cons}_{\mathsf{OLE}}^{\mathbb{F}_p}$ is programmable, we can re-define **Gen** by letting it take additional inputs (ρ_0, ρ_1) where $\rho_\sigma = (\mathbf{A}_\sigma^0, \mathbf{A}_\sigma^1, \boldsymbol{s}_\sigma, \boldsymbol{e}_\sigma)$, for $\sigma \in \{0, 1\}$. Note that ρ_σ corresponds two t-sparse elements s_σ, e_σ of $\mathcal{R}_k$ as well as an element $x_\sigma := a \cdot s_\sigma + e_\sigma \in \mathcal{R}_k$. According to Eq.(8), party P_σ can locally compute $(\mathsf{Tr}(\xi^{p^0} x_\sigma), \ldots, \mathsf{Tr}(\xi^{p^{k-1}} x_\sigma)) \in \mathbb{F}_p^{kN}$ from ρ_σ. This essentially proves the programmability. Correctness follows that of Theorem 5.1. The programmable security can be proved in a very similar way to that of Theorem 5.1. □

6 PCG for Authenticated Multiplication Triples

For the sake of generality, we give a generic PCG construction for authenticated triples over any field, which can be directly applied to the Boolean case.

Secret-sharing with MACs. We consider authenticated secret-sharing based on SPDZ MACs between m parties. To share a value $x \in \mathbb{F}_p$, when $\mathbb{F}_p$ is a small field, in particular, the binary field $\mathbb{F}_2$, the MAC key needs to be picked from a sufficiently large extension field. In this way, a sharing will be prevented from being opened incorrectly with overwhelming probability, via a MAC check method from [DKL+13]. The authenticated secret-sharing of $x \in \mathbb{F}_p$ is defined as:

$$[\![x]\!] = (\Delta_i, x_i, M_{x,i})_{i=1}^m \text{ such that } \sum_i x_i = x, \text{ and } \sum_i M_{x,i} = x \cdot \sum_i \Delta_i,$$

where $x_i \in \mathbb{F}_p$, $\Delta_i, M_{x,i} \in \mathbb{F}_{p^k}$, for each i. We call $\Delta := \sum_i \Delta_i \in \mathbb{F}_{p^k}$ the MAC key. Note that the MAC key shares Δ_i are fixed for every shared x. An *authenticated multiplication triple* for $\mathbb{F}_p$ is a tuple of random sharings $([\![x]\!], [\![y]\!], [\![z]\!])$, satisfying

$x, y \xleftarrow{\$} \mathbb{F}_p$ and $z = x \cdot y$. For the security of MPC online protocols, k is often selected as $k \log p = O(\lambda)$. For simplicity, we call it an authenticated Boolean triple if consider the case of $\mathbb{F}_p = \mathbb{F}_2$. We show an efficient PCG that outputs a batch of random authenticated triples over any field $\mathbb{F}_p$ for the two-party case.

Our PCG construction. We present a self-contained construction in $\mathsf{Cons}_{\mathsf{Auth-Triple}}$, which incorporates the PCG construction for authenticated triples over large fields from [BCG+20] and our $\mathsf{Cons}^{\mathbb{F}_p}_{\mathsf{OLE}}$ construction in Section 5. To begin with, we recall the notations. Let $\mathcal{R}_k$ denote $\mathbb{F}_{p^k}[\mathtt{X}_1, \ldots, \mathtt{X}_n]/(\mathtt{X}_1^d - 1, \ldots, \mathtt{X}_n^d - 1)$, where $d \mid p^k - 1$. We have $\mathcal{R}_k \simeq \mathbb{F}_{p^k}^N$, where $N = d^n$. We fix a basis of $(\xi^{p^0}, \xi^{p^1}, \ldots, \xi^{p^{k-1}})$ of $\mathbb{F}_{p^k} = \mathbb{F}_p(\xi)$. We assume QA-SD$(\mathcal{R}_k, t)$, in which $x_\sigma := a \cdot s_\sigma + e_\sigma$ is conjectured to be indistinguishable from uniformly random, for public $a \xleftarrow{\$} \mathcal{R}_k$, and random t-sparse elements s_σ, e_σ of $\mathcal{R}_k$, $\sigma \in \{0, 1\}$.

Recall that $\mathsf{Cons}^{\mathbb{F}_p}_{\mathsf{OLE}}$ essentially distributes $\xi^{p^s} x_\sigma, z_\sigma^{(s)} \in \mathcal{R}_k$ to party P_σ, with $\mathsf{Tr}(\xi^{p^s} x_0) * \mathsf{Tr}(\xi^{p^s} x_1) = \mathsf{Tr}(z_0^{(s)}) + \mathsf{Tr}(z_1^{(s)})$, for $\sigma \in \{0, 1\}$, $s \in [0, k-1]$. To get multiplication triples, it suffices to distribute additive shares of x_0, x_1 between the two parties, instead of having P_σ hold x_σ in the clear. This follows the fact that, for $\sigma \in \{0, 1\}$, $s \in [0, k-1]$,

$$\mathsf{Tr}(\xi^{p^s} \cdot [x_\sigma]_{\mathcal{R}_k}) = \mathsf{Tr}(\left[\xi^{p^s} \cdot x_\sigma\right]_{\mathcal{R}_k}) = \left[\mathsf{Tr}(\xi^{p^s} \cdot x_\sigma)\right]_{\mathbb{F}_p^N}.$$

Since $[x_\sigma]_{\mathcal{R}_k} := a \cdot [s_\sigma]_{\mathcal{R}_k} + [e_\sigma]_{\mathcal{R}_k}$ and s_σ, e_σ are t-sparse, we can use t-point SPFSS with domain $[0, N)$, range $\mathbb{F}_{p^k}$ to obtain the sharing succinctly. This leads to a seed size of $O\big(\lambda(kt^2 + t) \log N\big)$ bits.

The authentication part is more complicated and expensive. A direct solution is as follows, where the goal is to obtain:

$$(\left[\Delta \cdot \mathsf{Tr}(\xi^{p^s} x_0)\right]_{\mathbb{F}_{p^k}^N}, \left[\Delta \cdot \mathsf{Tr}(\xi^{p^s} x_1)\right]_{\mathbb{F}_{p^k}^N}, \left[\Delta \cdot \mathsf{Tr}(\xi^{p^s} x_0) \cdot \mathsf{Tr}(\xi^{p^s} x_1)\right]_{\mathbb{F}_{p^k}^N}),$$

for $j \in [0, k-1]$, where $\Delta \xleftarrow{\$} \mathbb{F}_{p^k}$ is fixed among triples. Due to that Tr is not $\mathbb{F}_{p^k}$-linear, the seed size would be roughly increased by a factor of $O(k)$. Concretely, for sharing the first two terms, it can be observed that

$$\begin{aligned} \Delta \cdot \mathsf{Tr}(\xi^{p^s} \cdot x_\sigma) &= \sum_{i \in [0, k-1]} \Delta \cdot \xi^{p^{s+i}} \cdot (a s_\sigma + e_\sigma)^{p^i} \\ &= \sum_{i \in [0, k-1]} \xi^{p^{s+i}} \cdot (a^{p^i} \Delta \cdot s_\sigma^{p^i} + \Delta \cdot e_\sigma^{p^i}). \end{aligned} \tag{9}$$

Hence, we have to distribute $\left[\Delta \cdot s_\sigma^{p^i}\right]_{\mathcal{R}_k}, \left[\Delta \cdot e_\sigma^{p^i}\right]_{\mathcal{R}_k}$ for all $i \in [0, k-1]$. This part would lead to a seed size of $O(\lambda k t \log N)$ bits via SPFSS. While for sharing

the last term, we have that

$$\begin{aligned}
&\Delta \cdot \mathsf{Tr}(x_0) \cdot \mathsf{Tr}(x_1) = \Delta \cdot (x_0^{p^{k-1}} + \ldots + x_0^{p^0}) \cdot (x_1^{p^{k-1}} + \ldots + x_1^{p^0}) \\
&= \sum_{i,j \in [0,k-1]} \Delta \cdot (as_0 + e_0)^{p^i} \cdot (as_1 + e_1)^{p^j} \\
&= \sum_{i,j \in [0,k-1]} \Delta \cdot (a^{p^i} s_0^{p^i} + e_0^{p^i}) \cdot (a^{p^j} s_1^{p^j} + e_1^{p^j}) \\
&= \sum_{i,j \in [0,k-1]} (a^{p^i+p^j} \Delta s_0^{p^i} s_1^{p^j} + a^{p^j} \Delta e_0^{p^i} s_1^{p^j} + a^{p^i} \Delta s_0^{p^i} e_1^{p^j} + \Delta e_0^{p^i} e_1^{p^j})
\end{aligned} \tag{10}$$

Hence we have to share $4k^2$ cross-terms, i.e. $\Delta s_0^{p^i} s_1^{p^j}, \Delta e_0^{p^i} s_1^{p^j}, \Delta s_0^{p^i} e_1^{p^j}, \Delta e_0^{p^i} e_1^{p^j}$, for all $i, j \in [0, k-1]$. Moreover, these cross-terms are t^2-sparse elements of $\mathcal{R}_k$ by Lemma 4.3. Similarly, by Theorem 4.3, we can extract kN correlations from these shares. Namely, fix a basis $(\xi^{p^0}, \xi^{p^1}, \ldots, \xi^{p^{k-1}})$, and for $s \in [0, k-1]$, we have

$$\begin{aligned}
&\Delta \cdot \mathsf{Tr}(\xi^{p^s} \cdot x_0) \cdot \mathsf{Tr}(\xi^{p^s} \cdot x_1) \\
&= \sum_{i,j \in [0,k-1]} \Delta \cdot \xi^{p^{s+i}} \cdot (as_0 + e_0)^{p^i} \cdot \xi^{p^{s+j}} \cdot (as_1 + e_1)^{p^j} \\
&= \sum_{i,j \in [0,k-1]} \xi^{p^s(p^i+p^j)} \cdot (a^{p^i+p^j} \Delta s_0^{p^i} s_1^{p^j} + a^{p^j} \Delta e_0^{p^i} s_1^{p^j} + a^{p^i} \Delta s_0^{p^i} e_1^{p^j} + \Delta e_0^{p^i} e_1^{p^j})
\end{aligned} \tag{11}$$

This part would incur a seed size of $O(\lambda t^2 k^2 \log N)$ via SPFSS. The $O(k)$ overhead of the above approach is quite large, since for security it is required that $k \log p = O(\lambda)$, i.e. $O(k) = O(\lambda)$. Here, we show a method that allows k to be an arbitrary integer ≥ 2, so that the overhead could be only a small constant.

Our main observation is that for any $k \geq 2$, the global key Δ can be sampled from $\mathbb{F}_{p^\eta}$ where $k \mid \eta$ and $\eta \log p = O(\lambda)$. This implies that there exists $\zeta \in \mathbb{F}_{p^\eta}$ s.t. $\mathbb{F}_{p^\eta} = \mathbb{F}_{p^k}[\zeta]$, and moreover $d \mid p^\eta - 1$ as long as $d \mid p^k - 1$. Then, by Theorem 4.4, the isomorphism $\phi_k : \mathcal{R}_k \to \mathbb{F}_{p^k}^N$ naturally induces $\phi_\eta : \mathcal{R}_\eta \to \mathbb{F}_{p^\eta}^N = \mathbb{F}_{p^k}^N(\zeta)$. Hence, for sharing values we can use a t^2-point SPFSS with domain $\mathbb{G} = \mathbb{Z}_d^n$, and range $\mathbb{F}_{p^k}$, while for sharing MACs, we can use an SPFSS', which is the same as SPFSS except that the range is $\mathbb{F}_{p^\eta}$. Putting all pieces together, the total seed size would be $O\big(\lambda(t^2k^2 + t^2k + tk + t) \log N\big) = O(\lambda^3 k^2 \log N)$ for kN authenticated triples over $\mathbb{F}_p$, where k is an arbitrary integer ≥ 2. We give a self-contained PCG construction for authenticated triples in $\mathsf{Cons}_{\mathsf{Auth-Triple}}$.

Construction 2: $\mathsf{Cons}_{\mathsf{Auth-Triple}}$

PARAMETER: Security parameter 1^λ, noise weight $t = t(\lambda)$, $d \mid p^k - 1$, $N = d^n$, η satisfying $p^\eta \geq 2^\lambda$ and $k \mid \eta$, $\mathcal{R}_k = \mathbb{F}_{p^k}[\mathtt{X}_1, \ldots, \mathtt{X}_n]/(\mathtt{X}_1^d - 1, \ldots, \mathtt{X}_n^d - 1)$, $\xi \in \mathbb{F}_{p^k}$ s.t. $\mathbb{F}_{p^k} = \mathbb{F}_p(\xi)$. An FSS scheme $(\mathsf{SPFSS.Gen}, \mathsf{SPFSS.FullEval})$ for sums of point

functions, with domain $[0, N)$[a] and range $\mathbb{F}_{p^k}$ or $\mathbb{F}_{p^\eta}$. We abuse ϕ as a fixed isomorphism map from $\mathcal{R}_k$ to $\mathbb{F}_{p^k}^N$, and from $\mathcal{R}_\eta$ to $\mathbb{F}_{p^\eta}^N$.
PUBLIC INPUT: A uniformly random $a \in \mathcal{R}_k$.
CORRELATION: After expansion, for $s \in [0, k)$, outputs authenticated triples $(\llbracket \boldsymbol{X}^{(s)} \rrbracket, \llbracket \boldsymbol{Y}^{(s)} \rrbracket, \llbracket \boldsymbol{Z}^{(s)} \rrbracket)$, where $\boldsymbol{X}^{(s)}, \boldsymbol{Y}^{(s)}, \boldsymbol{Z}^{(s)} \in \mathbb{F}_p^N$, satisfying $\boldsymbol{Z}^{(s)} = \boldsymbol{X}^{(s)} * \boldsymbol{Y}^{(s)}$, and MAC key shares $\Delta_0, \Delta_1 \in \mathbb{F}_{p^\eta}$.
Gen: On input 1^λ:

1. Sample $\Delta_0, \Delta_1 \xleftarrow{\$} \mathbb{F}_{p^\eta}$, and compute $\Delta := \Delta_0 + \Delta_1$.
2. For $\sigma \in \{0, 1\}$, sample random vectors $\mathbf{A}_\sigma^0, \mathbf{A}_\sigma^1 \leftarrow [0, N)^t$, $\boldsymbol{s}_\sigma, \boldsymbol{e}_\sigma \leftarrow (\mathbb{F}_{p^k}^*)^t$. Define elements of $\mathcal{R}_k$:
$$s_\sigma = \sum_{l=1}^{t} \boldsymbol{s}_\sigma[l] \cdot \mathbf{A}_\sigma^0[l], \quad e_\sigma = \sum_{l=1}^{t} \boldsymbol{e}_\sigma[l] \cdot \mathbf{A}_\sigma^1[l].$$
3. Sample FSS keys (sharing of $\Delta \cdot \boldsymbol{Z}^{(s)}$) according to Eq.(10), for $i, j \in [0, k)$:
$(K_0^{4(ki+j)}, K_1^{4(ki+j)}) \xleftarrow{\$} \mathsf{SPFSS.Gen}(1^\lambda, (\mathbf{A}_0^0)^{p^i} \boxplus (\mathbf{A}_1^0)^{p^j}, \Delta \cdot (\boldsymbol{s}_0)^{p^i} \otimes (\boldsymbol{s}_1)^{p^j})$,
$(K_0^{4(ki+j)+1}, K_1^{4(ki+j)+1}) \xleftarrow{\$} \mathsf{SPFSS.Gen}(1^\lambda, (\mathbf{A}_0^1)^{p^i} \boxplus (\mathbf{A}_1^0)^{p^j}, \Delta \cdot (\boldsymbol{e}_0)^{p^i} \otimes (\boldsymbol{s}_1)^{p^j})$,
$(K_0^{4(ki+j)+2}, K_1^{4(ki+j)+2}) \xleftarrow{\$} \mathsf{SPFSS.Gen}(1^\lambda, (\mathbf{A}_0^0)^{p^i} \boxplus (\mathbf{A}_1^1)^{p^j}, \Delta \cdot (\boldsymbol{s}_0)^{p^i} \otimes (\boldsymbol{e}_1)^{p^j})$,
$(K_0^{4(ki+j)+3}, K_1^{4(ki+j)+3}) \xleftarrow{\$} \mathsf{SPFSS.Gen}(1^\lambda, (\mathbf{A}_0^1)^{p^i} \boxplus (\mathbf{A}_1^1)^{p^j}, \Delta \cdot (\boldsymbol{e}_0)^{p^i} \otimes (\boldsymbol{e}_1)^{p^j})$,
4. Sample FSS keys (sharing of $\boldsymbol{Z}^{(s)}$) according to Eq.(7), for $i \in [0, k)$:
$(K_0^{4k^2+4i}, K_1^{4k^2+4i}) \xleftarrow{\$} \mathsf{SPFSS.Gen}(1^\lambda, \mathbf{A}_0^0 \boxplus (\mathbf{A}_1^0)^{p^i}, \boldsymbol{s}_0 \otimes (\boldsymbol{s}_1)^{p^i})$,
$(K_0^{4k^2+4i+1}, K_1^{4k^2+4i+1}) \xleftarrow{\$} \mathsf{SPFSS.Gen}(1^\lambda, \mathbf{A}_0^1 \boxplus (\mathbf{A}_1^0)^{p^i}, \boldsymbol{e}_0 \otimes (\boldsymbol{s}_1)^{p^i})$,
$(K_0^{4k^2+4i+2}, K_1^{4k^2+4i+2}) \xleftarrow{\$} \mathsf{SPFSS.Gen}(1^\lambda, \mathbf{A}_0^0 \boxplus (\mathbf{A}_1^1)^{p^i}, \boldsymbol{s}_0 \otimes (\boldsymbol{e}_1)^{p^i})$,
$(K_0^{4k^2+4i+3}, K_1^{4k^2+4i+3}) \xleftarrow{\$} \mathsf{SPFSS.Gen}(1^\lambda, \mathbf{A}_0^1 \boxplus (\mathbf{A}_1^1)^{p^i}, \boldsymbol{e}_0 \otimes (\boldsymbol{e}_1)^{p^i})$,
5. Sample FSS keys (sharing of $\Delta \cdot \boldsymbol{X}^{(s)}, \Delta \cdot \boldsymbol{Y}^{(s)}$) according to Eq.(9), for $i \in [0, k)$:
$(K_0^{4(k^2+k+i)}, K_1^{4(k^2+k+i)}) \xleftarrow{\$} \mathsf{SPFSS.Gen}(1^\lambda, \mathbf{A}_0^0, \Delta \cdot (\boldsymbol{s}_0)^{p^i})$,
$(K_0^{4(k^2+k+i)+1}, K_1^{4(k^2+k+i)+1}) \xleftarrow{\$} \mathsf{SPFSS.Gen}(1^\lambda, \mathbf{A}_0^1, \Delta \cdot (\boldsymbol{e}_0)^{p^i})$,
$(K_0^{4(k^2+k+i)+2}, K_1^{4(k^2+k+i)+2}) \xleftarrow{\$} \mathsf{SPFSS.Gen}(1^\lambda, \mathbf{A}_1^0, \Delta \cdot (\boldsymbol{s}_1)^{p^i})$,
$(K_0^{4(k^2+k+i)+3}, K_1^{4(k^2+k+i)+3}) \xleftarrow{\$} \mathsf{SPFSS.Gen}(1^\lambda, \mathbf{A}_1^1, \Delta \cdot (\boldsymbol{e}_1)^{p^i})$,
6. Sample FSS keys (sharing of $\boldsymbol{X}^{(s)}, \boldsymbol{Y}^{(s)}$) as follows:
$(K_0^{4k^2+8k}, K_1^{4k^2+8k}) \xleftarrow{\$} \mathsf{SPFSS.Gen}(1^\lambda, \mathbf{A}_0^0, \boldsymbol{s}_0)$,
$(K_0^{4k^2+8k+1}, K_1^{4k^2+8k+1}) \xleftarrow{\$} \mathsf{SPFSS.Gen}(1^\lambda, \mathbf{A}_0^1, \boldsymbol{e}_0)$,
$(K_0^{4k^2+8k+2}, K_1^{4k^2+8k+2}) \xleftarrow{\$} \mathsf{SPFSS.Gen}(1^\lambda, \mathbf{A}_1^0, \boldsymbol{s}_1)$,
$(K_0^{4k^2+8k+3}, K_1^{4k^2+8k+3}) \xleftarrow{\$} \mathsf{SPFSS.Gen}(1^\lambda, \mathbf{A}_1^1, \boldsymbol{e}_1)$,
7. For $\sigma \in \{0, 1\}$, let $\mathbf{k}_\sigma = \left(\Delta_\sigma, (K_\sigma^i)_{i \in [0, 4k^2+8k+3]}\right)$.
8. Output $(\mathbf{k}_0, \mathbf{k}_1)$.

Expand: On input $(\sigma, \mathbf{k}_\sigma)$:

1. Parse $\mathbf{k}_\sigma$ as $\Big(\Delta_\sigma, (K_\sigma^{4(ki+j)})_{i,j\in[0,k)}, (K_\sigma^{4k^2+i})_{i\in[0,8k)}, (K_\sigma^{4k^2+8k+i})_{i\in[0,3)}\Big)$.
2. For each $i \in [0, 4k^2)$, compute $u_{\sigma,i} \leftarrow \mathsf{SPFSS.FullEval}(\sigma, K_\sigma^i)$, viewed as $\mathcal{R}_\eta$ elements. And for each $i \in [0, 8k)$, compute $v_{\sigma,i} \leftarrow \mathsf{SPFSS.FullEval}(\sigma, K_\sigma^{4k^2+i})$. View $v_{\sigma,i}$ as $\mathcal{R}_k$ elements for $i \in [0, 4k)$, and as $\mathcal{R}_\eta$ elements for $i \in [4k, 8k)$. Finally for each $i \in [0, 4)$, compute $w_{\sigma,i} \leftarrow \mathsf{SPFSS.FullEval}(\sigma, K_\sigma^{4k^2+8k+i})$, viewed as $\mathcal{R}_k$ elements.
3. For each $s \in [0, k-1]$, according to Eq.(8), compute

$$\boldsymbol{Z}_\sigma^{(s)} := \sum_{i=0}^{k-1} \mathsf{Tr}\big(\xi^{p^s+p^{s+i}} \cdot (a^{p^i+1} v_{\sigma,4i} + a^{p^i} v_{\sigma,4i+1} + a v_{\sigma,4i+2} + v_{\sigma,4i+3})\big).$$

   It holds that $\boldsymbol{Z}_0^{(s)} + \boldsymbol{Z}_1^{(s)} = \mathsf{Tr}\big(\xi^{p^s} \cdot (as_0 + e_0)\big) * \mathsf{Tr}\big(\xi^{p^s} \cdot (as_1 + e_1)\big)$.
4. For each $s \in [0, k-1]$, according to Eq.(10), compute

$$\boldsymbol{M}_{Z,\sigma}^{(s)} := \phi\Big(\sum_{i=0}^{k-1}\sum_{j=0}^{k-1} \xi^{p^s(p^i+p^j)} \cdot \big(a^{p^i+p^j} u_{\sigma,4(ki+j)} + a^{p^j} u_{\sigma,4(ki+j)+1} + a^{p^i} u_{\sigma,4(ki+j)+2} + u_{\sigma,4(ki+j)+3}\big)\Big).$$

   It holds that $\boldsymbol{M}_{Z,0}^{(s)} + \boldsymbol{M}_{Z,1}^{(s)} = \Delta \cdot \mathsf{Tr}\big(\xi^{p^s} \cdot (as_0 + e_0)\big) * \mathsf{Tr}\big(\xi^{p^s} \cdot (as_1 + e_1)\big)$.
5. For each $s \in [0, k-1]$, compute the following

$$\boldsymbol{X}_\sigma^{(s)} := \mathsf{Tr}\big(\xi^{p^s} \cdot (a w_{\sigma,0} + w_{\sigma,1})\big), \quad \boldsymbol{Y}_\sigma^{(s)} := \mathsf{Tr}\big(\xi^{p^s} \cdot (a w_{\sigma,2} + w_{\sigma,3})\big).$$

   It holds that $\boldsymbol{X}_0^{(s)} + \boldsymbol{X}_1^{(s)} = \mathsf{Tr}\big(\xi^{p^s} \cdot (as_0 + e_0)\big)$, and $\boldsymbol{Y}_0^{(s)} + \boldsymbol{Y}_1^{(s)} = \mathsf{Tr}\big(\xi^{p^s} \cdot (as_1 + e_1)\big)$.
6. For each $s \in [0, k-1]$, compute the following

$$\boldsymbol{M}_{X,\sigma}^{(s)} := \phi\big(\sum_{i=0}^{k-1} \xi^{p^{s+i}} \cdot (a^{p^i} v_{\sigma,4k+4i} + v_{\sigma,4k+4i+1})\big),$$
$$\boldsymbol{M}_{Y,\sigma}^{(s)} := \phi\big(\sum_{i=0}^{k-1} \xi^{p^{s+i}} \cdot (a^{p^i} v_{\sigma,4k+4i+2} + v_{\sigma,4k+4i+3})\big).$$

   It holds that $\boldsymbol{M}_{X,0}^{(s)} + \boldsymbol{M}_{X,1}^{(s)} = \Delta \cdot \mathsf{Tr}\big(\xi^{p^s} \cdot (as_0 + e_0)\big)$, and $\boldsymbol{M}_{Y,0}^{(s)} + \boldsymbol{M}_{Y,1}^{(s)} = \Delta \cdot \mathsf{Tr}\big(\xi^{p^s} \cdot (as_1 + e_1)\big)$.
7. Output $\Delta_\sigma \in \mathbb{F}_{p^\lambda}$, and $(\boldsymbol{X}_\sigma^{(s)}, \boldsymbol{Y}_\sigma^{(s)}, \boldsymbol{Z}_\sigma^{(s)}, \boldsymbol{M}_{X,\sigma}^{(s)}, \boldsymbol{M}_{Y,\sigma}^{(s)}, \boldsymbol{M}_{Z,\sigma}^{(s)}) \in \mathbb{F}_p^{3N} \times \mathbb{F}_{p^\eta}^{3N}$, for $s \in [0, k)$.

[a] Each $j \in [0, N)$ corresponds to a basis $\prod_{i=1}^n \mathtt{x}_i^{j_i}$ of $\mathcal{R}_k$, where $j_i \in [0, p^k - 1)$. The Addition on $[0, N)$ is actually the addition on $\mathbb{G} := \mathbb{Z}_d^n$.

Theorem 6.1. *Assume a secure* SPFSS *for sums of point functions and QA-SD($\mathcal{R}_k, t$) is hard. Then there exists a PCG construction* $\mathsf{Cons}_{\mathsf{Auth-Triple}}$ *that gener-*

ates authenticated multiplication triple correlations over $\mathbb{F}_p$. If the SPFSS *is based on a* $\mathsf{PRG} : \{0,1\}^\lambda \to \{0,1\}^{2\lambda+2}$ *via the* PRG*-based construction from [BGI16], to produce kN authenticated multiplication triples over $\mathbb{F}_p$, we have that:*

- *Each party has seed size at most:* $4(k^2t^2+kt^2+kt+t)\big((\log N-\log t+1)(\lambda+2)+\lambda\big)+4\big(\eta(k^2t^2+kt)+k(kt^2+t)\big)\log p+\eta\log p$ *bits.*
- *Suppose k is a small constant, then computations of* **Expand** *are dominated by* $4k^2tN\cdot(2+\left\lceil\frac{\eta\log p}{\lambda}\right\rceil)$ *PRG calls and* $O(k^3N\log N)$ *operations over* $\mathbb{F}_{p^\eta}$.

7 PCG for Other Correlations and Applications

In this section, we show more applications of our approach from trace. In Section 7.1, we generalize our PCG for OLE to the multiparty setting by utilizing the programmability. In Section 7.2&7.3, we consider other useful correlations, such as matrix multiplication triples and OT. To this end, we construct new PCGs by extending our trace approach. Compared to previous PCG constructions [BCG+19a, BCG+19b, BCG+20], our PCG for matrix triples has smaller seed size, and our PCG for OT does not require correlation-robust hash functions.

7.1 MPC with Preprocessing

Multiplication triples for preprocessing MPC. Recall that our PCG construction $\mathsf{Cons}_{\mathsf{OLE}}^{\mathbb{F}_p}$satisfies "programmability" (Theorem 5.2), which allows to generate multi-party correlations as shown in [BCG+19b]. Here we consider the correlation of m-party multiplication triples, also known as Beaver triple [Bea91]. Concretely, the goal is to distribute additives shares of x, y, z to the parties,

$$([x],[y],[z]) = (x_i, y_i, z_i)_{i=1}^m \ \ s.t.\ (\sum_{i=1}^m x_i)\cdot(\sum_{i=1}^m y_i) = (\sum_{i=1}^m z_i).$$

A standard solution is to distribute each party P_i with x_i, y_i and $\{u_{i,j}, v_{i,j}\}_{j\neq i}$, such that $u_{i,j}+u_{j,i} = x_i\cdot y_j$, and $v_{i,j}+v_{j,i} = y_i\cdot x_j$. Then P_i can locally compute $z_i := x_i\cdot y_i + \sum_{j\neq i}(u_{i,j}+v_{i,j})$. It holds that

$$\begin{aligned}\sum_{i=1}^m z_i &= \sum_{i=1}^m \big(x_i\cdot y_i + \sum_{j\neq i}(u_{i,j}+v_{i,j})\big) = \sum_{i=1}^m \big(x_i\cdot y_i + \sum_{j>i}(u_{i,j}+u_{j,i}+v_{i,j}+v_{j,i})\big)\\ &= \sum_{i=1}^m \big(x_i\cdot y_i + \sum_{j>i}(x_i\cdot y_j + y_i\cdot x_j)\big) = (\sum_{i=1}^m x_i)\cdot(\sum_{i=1}^m y_i).\end{aligned}$$

The programmability allows x_i, y_i to be "reused" among $\mathsf{Cons}_{\mathsf{OLE}}^{\mathbb{F}_p}$ calls on distributing additives shares of $x_i\cdot y_j$, $y_i\cdot x_j$ between P_i and P_j, for all $j\neq i$. Hence, our $\mathsf{Cons}_{\mathsf{OLE}}^{\mathbb{F}_p}$ implies a PCG for multi-party multiplication triples over any field $\mathbb{F}_p$.

Theorem 7.1. *Assume a secure* FSS *scheme* SPFSS *for sums of point functions and QA-SD($\mathcal{R}_k, t$) is hard. Then there exists a PCG that generates Beaver triples over $\mathbb{F}_p$. If the* SPFSS *is based on a* PRG : $\{0,1\}^\lambda \to \{0,1\}^{2\lambda+2}$ *via the* PRG*-based construction from [BGI16], for generating kN m-party multiplication correlations over $\mathbb{F}_p$, we have that:*

- *Each party's seed has maximum size around:* $8(m-1)kt^2\big((\log N - \log t + 1)(\lambda+2) + \lambda + k\log p\big) + 4t(\log N + k\log p)$ *bits.*
- *Each party runs* ***Expand*** *of* $\mathsf{Cons}_{\mathsf{OLE}}^{\mathbb{F}_p}$ *for* $2(m-1)$ *times, and does additional* $O(mk^2N)$ *operations over* $\mathbb{F}_p$.

Circuit-dependent MPC preprocessing. Circuit-dependent preprocessing extends the standard Beaver's circuit randomization technique with multiplication triples [Bea91], which in general allows to reduce the MPC online communication by half. The intuitive idea is to preprocess multiplications according to the circuit topology, so that in the online phase just one opening is required per multiplication gate, instead of two when using multiplication triples.

Circuit-dependent preprocessing has been investigated in recent works [DNNR17, Cou19, BNO19, EGPS22, EGP+23]. Previous works [BCG+20, BCCD23] show that PCGs offer a significant saving for circuit-dependent preprocessing in the batch setting[4], which cost only $O(|\mathcal{C}|\log B)$ communication for computing B copies of the same circuit $\mathcal{C}$. However, due to the inherent field restriction of their PCG constructions, neither of them can be used to efficiently support the Boolean circuit case. Using our approach from trace, we are able to do efficient circuit-dependent preprocessing for any field. Our results are summarized as follows.

Theorem 7.2. *Assume a secure* FSS *scheme* SPFSS *for sums of point functions and QA-SD($\mathcal{R}_k, t$) is hard. Suppose the* SPFSS *is based on a* PRG : $\{0,1\}^\lambda \to \{0,1\}^{2\lambda+2}$ *via the* PRG*-based construction from [BGI16]. Then there exists a PCG-based m-party preprocessing construction that generates circuit-dependent correlations for kN evaluations of a circuit $\mathcal{C}$ over any field $\mathbb{F}_p$ with following:*

- *Total communication is* $O(m^2kt^2\lambda|\mathcal{C}|\log N)$ *bits.*
- *Each party makes* $O(mkt|\mathcal{C}|N)$ PRG *calls, and has* $O(mk^2|\mathcal{C}|N\log N)$ *operations in* $\mathbb{F}_{p^k}$.

Proof. The proof directly follows that of [BCG+20, BCCD23], and for completeness, we sketch the main idea. For each evaluation, the goal is to assign each wire w a fresh mask r_w and each multiplication gate with input wires u, v a fresh mask $s_{u,v}$, and distribute additive shares of these masks to the parties. The masks are designed as follows.

- *if w is an input wire,* $r_w \xleftarrow{\$} \mathbb{F}_p$.

[4] We remark that the batch setting seems necessary, due to the fact that the correlated randomness depends on the general circuit topology, so that it cannot be compressed beyond the description of the circuit.

- *if w is the output wire of a multiplication gate, $r_w \xleftarrow{\$} \mathbb{F}_p$.*
- *if w is the output wire of an addition gate with input wire u and v, set $r_w = r_u + r_v$.*
- *for each multiplication gate, assign a value $s_{u,v}$ s.t. on input wire u and v, $s_{u,v} = r_u \cdot r_v$.*

For sharing $s_{u,v}$, by the standard transformation of m-party multiplication triples from 2-party OLE, it suffices to use our programmable $\mathsf{Cons}_{\mathsf{OLE}}^{\mathbb{F}_p}$ to generate seeds for the $m(m-1)/2$ pairs of parties. Methods of sharing masks r_w (of an input wire or the output wire of a multiplication gate) are directly implied by $\mathsf{Cons}_{\mathsf{OLE}}^{\mathbb{F}_p}$. By Theorem 5.1, we complete the proof. □

We remark that our $\mathsf{Cons}_{\mathsf{Auth-Triple}}$ can also be extended to do circuit-dependent preprocessing in an authentication sense, e.g., used for [BNO19] in the two-party setting. In this case, we also require the batch setting, and the offline communication is $O(\lambda k^2 t^2 |\mathcal{C}| \log N)$ for kN executions of $\mathcal{C}$ over any field.

7.2 PCG for Matrix Multiplication Triples

In this section, we show an efficient PCG construction for generating matrix multiplication triples over any field $\mathbb{F}_p$.

Subfield OLE and Correlated Subfield OLE. To start with, we introduce two new notions of subfield OLE and correlated subfield OLE. Subfield OLE has many similarities with existing widely used correlations, such as OT, OLE, VOLE, etc. More concisely, in a subfield OLE correlation, party P_0 holds $\mathbf{b} \in \mathbb{F}_p^N, \mathbf{z}_0 \in \mathbb{F}_{p^k}^N$, while party P_1 holds $\mathbf{x}, \mathbf{z}_1 \in \mathbb{F}_{p^k}^N$, such that

$$\mathbf{b} * \mathbf{x} = \mathbf{z}_0 + \mathbf{z}_1.$$

On the one hand, subfield OLE is a natural generalization of subfield VOLE, in the sense that in subfield VOLE, all entries of $\mathbf{x}$ are identical to some $x \in \mathbb{F}_{p^k}$. On the other hand, by fixing a basis $(\xi^0, \xi^1, \ldots, \xi^{k-1})$ and viewing $\mathbb{F}_{p^k} = \mathbb{F}_p(\xi)$ as a vector space over $\mathbb{F}_p$, the above subfield OLE defines N instances of VOLE over $\mathbb{F}_p$ with a fixed length k, given by $\mathbf{b}[s] \cdot \mathbf{x}[s] = \mathbf{z}_0[s] + \mathbf{z}_1[s]$, for each $s \in [0, N)$. We will present efficient PCG constructions for subfield OLE in Section 7.3.

Correlated subfield OLE is a variant of subfield OLE, in which party P_0 holds $\mathbf{b}^{(j)} \in \mathbb{F}_p^N, \mathbf{z}_0^{(j)} \in \mathbb{F}_{p^k}^N$, while party P_1 holds $\mathbf{x}, \mathbf{z}_1^{(j)} \in \mathbb{F}_{p^k}^N$, such that

$$\mathbf{b}^{(j)} * \mathbf{x} = \mathbf{z}_0^{(j)} + \mathbf{z}_1^{(j)},$$

for all $j \in [1, \ell]$. Similarly, by fixing a basis $(\xi^0, \xi^1, \ldots, \xi^{k-1})$ of $\mathbb{F}_{p^k} = \mathbb{F}_p(\xi)$ over $\mathbb{F}_p$, the above correlated subfield OLE correlation defines ℓN instances of VOLE over $\mathbb{F}_p$, denoted by $\mathbf{b}^{(j)}[s] \cdot \vec{\mathbf{x}}[s] = \vec{\mathbf{z}}_0^{(j)}[s] + \vec{\mathbf{z}}_1^{(j)}[s]$, where $\mathbf{b}^{(j)}[s] \in \mathbb{F}_p$, and $\vec{\mathbf{x}}[s], \vec{\mathbf{z}}_0^{(j)}[s], \vec{\mathbf{z}}_1^{(j)}[s] \in \mathbb{F}_p^k$, for each $j \in [1, \ell], s \in [0, N)$. We will show how to build efficient PCG constructions for correlated subfield OLE later in this section. In particular, our construction works for any $\ell \leq k$.

Matrix Multiplication Triples from Correlated Subfield OLE. We show that PCG for correlated subfield OLE directly implies PCG for matrix multiplication triples. Let $\mathbb{F}_p$ be a field and w.l.o.g., we consider multiplications of $k \times k$ matrices. Specifically, the goal is to to produce N random triples of $([A], [B], [C])$, where $A, B, C \in \mathbb{F}_p^{k\times k}$ satisfy $AB = C$. To begin with, we show how to obtain $[C]$ from VOLE.

Denote entries of A by $a_{i,j}$, where $i, j \in [1, k]$. Let $\vec{A}_j$ denote the j-th column of A (same for B, C), for $j \in [1, k]$. To obtain $\left[\vec{C}_j\right]$ of $C = AB$, it suffices to distribute additive shares of the following matrix:

$$M = \begin{pmatrix} \vec{A}_1 b_{1,1} & \vec{A}_2 b_{2,1} & \dots & \vec{A}_k b_{k,1} \\ \vec{A}_1 b_{1,2} & \vec{A}_2 b_{2,2} & \dots & \vec{A}_k b_{k,2} \\ \vdots & \vdots & \vdots & \vdots \\ \vec{A}_1 b_{1,k} & \vec{A}_2 b_{2,k} & \dots & \vec{A}_k b_{k,k} \end{pmatrix},$$

which corresponds to kN VOLE correlations[5], with a fixed length k. A direct observation is that each $\vec{A}_i$ would be reused for k times. The authors of [LXYZ24] exploit this observation and show that to obtain N matrix triples, one can "repeat" for kN times a programmable PCG for generating VOLE of length k. Their approach leads to a PCG seed size of $O_\lambda(kNt \log k)$, where t is the noise weight of underlying LPN assumption.

As correlated subfield OLE can be locally transformed into VOLEs, we present a more efficient approach to generating a large number of matrix triples. Recall we want to obtain N random triples of $(\left[A^{(s)}\right], \left[B^{(s)}\right], \left[C^{(s)}\right])$, for each $s \in [0, N)$. Note that our PCG construction for correlated subfield OLE is able to produce

$$\mathbf{b}^{(j)} \cdot \mathbf{x} = \mathbf{z}_0^{(j)} + \mathbf{z}_1^{(j)},$$

where $\mathbf{b}^{(j)} \in \mathbb{F}_p^{kN}$ and $\mathbf{x}, \mathbf{z}_0, \mathbf{z}_1 \in \mathbb{F}_{p^k}^{kN}$, for $j \in [1, k]$.

For $s \in [0, N), i \in [1, k]$, we can view $\mathbf{x}[ks + i] \in \mathbb{F}_{p^k}$ as an element of $\mathbb{F}_p^k$, and denote it by $\vec{A}_i^{(s)}$, and denote $\mathbf{b}^{(j)}[ks + i]$ by $b_{i,j}^{(s)}$ for each $j \in [1, k]$. Note that we have already obtained $\left[\vec{A}_i^{(s)} b_{i,j}^{(s)}\right]$ for all $i, j \in [1, k], s \in [0, N)$. This essentially completes the key step of $\mathsf{PCG.Expand}$ for matrix triples. It remains to distribute $\left[A^{(s)}\right], \left[B^{(s)}\right]$ for each $s \in [0, N)$ between $\mathrm{P}_0, \mathrm{P}_1$ (rather than directly send $\mathbf{b}^{(j)}$ to P_0, and $\mathbf{x}$ to P_1 as in correlated subfield OLE). Jumping ahead, in our PCG construction, all $\mathbf{b}^{(j)}$, $j \in [1, k]$ are determined by a QA-SD sample $x_0 := as_0 + e_0$ while $\mathbf{x}$ is determined by $x_1 := as_1 + e_1$. Suppose s_0, e_0, s_1, e_1 are t-sparse, then it suffice to employ a SPFSS scheme with domain $[0, N)$ and range $\mathbb{F}_{p^k}$ to efficiently distribute additive shares of x_0, x_1. Hence assuming the hardness of QA-SD$(\mathcal{R}_k, t)$, the seed size is $O_\lambda\big(4(kt^2 + t)(k + \log kN)\big)$, where $N \le \lfloor (p^k - 1)^n / k \rfloor$, and $\mathcal{R}_k = \mathbb{F}_{p^k}[\mathtt{X}_1, \dots, \mathtt{X}_n]/(\mathtt{X}_1^{p^k-1} - 1, \dots, \mathtt{X}_n^{p^k-1} - 1)$.

[5] We remark that for matrix triples, VOLE is defined in the sense that $\mathrm{P}_0, \mathrm{P}_1$ hold additive shares of each $\vec{A}_j$, $b_{i,j}$ instead of P_0 holding $\vec{A}_j$, P_1 holding $b_{i,j}$ in the clear.

We remark here that [BCG+20] also proposes a PCG for matrix triples from Ring-LPN. Briefly, they start with a PCG for inner products, and rely on the ring $\mathbb{F}_p[\mathtt{X}]/(f(\mathtt{X}))$ with $f(\mathtt{X}) = \prod_{i=1}^{N}(\mathtt{X} - \alpha_i)^k$, where α_i are distinct elements of $\mathbb{F}_p$. The seed size of their PCG is $O_\lambda(4k^2t^2 \log N)$, where t is the noise weight of the underlying Ring-LPN assumption. As indicated above, this approach has a undesirable restriction on the field size, namely, $|\mathbb{F}_p| > N$.

We remark that as QA-SD assumptions essentially build upon on multi-variate polynomial rings with fully reducible $f(\mathtt{X}_i)$, the PCG construction of [BCCD23] does not enjoy the above optimization for inner products. Hence the seed size would be $O_\lambda(4k^3t^2 \log N)$, if using their PCG for OLE ($|\mathbb{F}_p| > 2$).

PCG for Matrix Multiplication Triples. As indicated above, it suffices to consider distributing correlated subfield OLE correlations, and our PCG construction also exploits algebraic properties of trace. We start with a subfield OLE construction. Observing that subfield OLE is not a symmetric correlation, in which P_0's multiplier $\mathbf{b} \in \mathbb{F}_p^N$, while P_1's multiplier $\mathbf{x}$ is over $\mathbb{F}_{p^k}$, the intuitive idea is to apply trace only on the P_0 side.

We first recall the notations. Let $\mathcal{R}_k = \mathbb{F}_{p^k}[\mathtt{X}_1, \ldots, \mathtt{X}_n]/(\mathtt{X}_1^{p^k-1} - 1, \ldots, \mathtt{X}_n^{p^k-1} - 1)$. Let $a \xleftarrow{\$} \mathcal{R}_k$ be the public input, and s_σ, e_σ be random t-sparse elements of $\mathcal{R}_k$, for $\sigma \in \{0, 1\}$. The hardness of QA-SD$(\mathcal{R}_k, t)$ implies that $x_\sigma := a \cdot s_\sigma + e_\sigma \in \mathcal{R}_k$ is indistinguishable from uniform. Let $\phi_k : \mathcal{R}_k \to \mathbb{F}_{p^k}^N$ be a bijection, where $N = (p^k - 1)^n$. Then to obtain subfield OLE correlations, we can use SPFSS to additively share $(x_0 + x_0^{p^1} + \cdots + x_0^{p^{k-1}}) \cdot x_1$ among $\mathrm{P}_0, \mathrm{P}_1$. Denote the additive shares by α, β, we have that

$$\mathsf{Tr}(x_0) \cdot x_1 = \alpha + \beta,$$

which are N subfield OLE correlations as desired. By Lemma 4.3, we have that

$$\begin{aligned}
(\sum_{i=0}^{k-1} x_0^{p^i}) \cdot x_1 &= \big(\sum_{i=0}^{k-1} (as_0 + e_0)^{p^i}\big) \cdot (as_1 + e_1) \\
&= \big(\sum_{i=0}^{k-1} (a^{p^i} s_0^{p^i} + e_0^{p^i})\big) \cdot (as_1 + e_1) \\
&= \sum_{i=0}^{k-1} (a^{p^i+1} s_0^{p^i} s_1 + a^{p^i} s_0^{p^i} e_1 + a e_0^{p^i} s_1 + e_0^{p^i} e_1).
\end{aligned} \tag{12}$$

Hence, it suffices to invoke SPFSS with domain $[0, N)$ and range $\mathbb{F}_{p^k}$ for sharing these $4k$ t^2-sparse elements of $\mathcal{R}_k$, and the total seed size would be $O\big(4kt^2(k + \log N)\big)$ bits. We remark that such PCG construction is programmable, and to obtain correlated subfield OLE, one can "repeat" PCG for subfield OLE for ℓ times. However, this naive approach leads to an $O(\ell)$ overhead.

Instead, we can extract at most kN correlated subfield OLE correlations by Theorem 4.3. By fixing a basis $(1, \xi^p, \ldots, \xi^{p^{k-1}})$ of $\mathbb{F}_{p^k} = \mathbb{F}_p(\xi)$, the extracted

correlations can be computed from

$$\Big(\sum_{i=0}^{k-1}(\xi^{p^j}x_0)^{p^i}\Big)\cdot x_1 = \sum_{i=0}^{k-1}\xi^{p^{i+j}}(a^{p^i+1}s_0^{p^i}s_1 + a^{p^i}s_0^{p^i}e_1 + ae_0^{p^i}s_1 + e_0^{p^i}e_1), \quad (13)$$

for $j \in [0, k-1]$. More specifically, P_0 computes $\mathbf{b}^{(j)} := \mathsf{Tr}(\xi^{p^{j-1}}x_0)$ for $j \in [1, k]$, while P_1 computes $\mathbf{x} := \phi_k(x_1)$. As for $\mathbf{z}_\sigma^{(j)}$, $\sigma \in \{0,1\}$, they can be locally computed according to Eq.(13). This completes the PCG construction for correlated subfield OLE. As for matrix multiplication triples, it suffices to let $\mathrm{P}_0, \mathrm{P}_1$ obtain additive shares of x_0, x_1 instead of P_σ receiving x_σ in the clear, for $\sigma \in \{0,1\}$. This can be done by sharing t-sparse elements s_0, e_0, s_1, e_1 of $\mathcal{R}_k$ via a SPFSS scheme. We present a PCG construction for matrix multiplication triples in $\mathsf{Cons}_{\mathsf{MatrixTriple}}$.

Construction 3: $\mathsf{Cons}_{\mathsf{MatrixTriple}}$

PARAMETER: Security parameter λ, noise weight $t = t(\lambda)$, $N = (p^k-1)^n$, $\mathcal{R}_k = \mathbb{F}_{p^k}[\mathtt{X}_1, \ldots, \mathtt{X}_n]/(\mathtt{X}_1^{p^k-1}-1, \ldots, \mathtt{X}_n^{p^k-1}-1)$, $\xi \in \mathbb{F}_{p^k}$ s.t. $\mathbb{F}_{p^k} = \mathbb{F}_p(\xi)$. An SPFSS scheme $(\mathsf{SPFSS.Gen}, \mathsf{SPFSS.FullEval})$ for sums of t^2 point functions, with domain $[0,N)$[a] and range $\mathbb{F}_{p^k}$.
PUBLIC INPUT: A uniformly random $a \in \mathcal{R}_k$.
CORRELATION: After expansion, outputs $(\left[A^{(s)}\right], \left[B^{(s)}\right], \left[C^{(s)}\right])$ with $A^{(s)}, B^{(s)}, C^{(s)} \in \mathbb{F}_p^{k\times k}$ s.t. $C^{(s)} = A^{(s)}B^{(s)}$, for $s \in [0, N')$, $N' = \lfloor N/k \rfloor$.
Gen: On input 1^λ:

1. For $\sigma \in \{0,1\}$, sample random vectors $\mathbf{A}_\sigma^0, \mathbf{A}_\sigma^1 \leftarrow [0,N)^t$, $\boldsymbol{s}_\sigma, \boldsymbol{e}_\sigma \leftarrow (\mathbb{F}_{p^k}^*)^t$.
2. Sample FSS keys (referring to $C^{(s)}$) according to Eq.(12), for each $i \in [0,k)$:
   $(K_0^{4i}, K_1^{4i}) \xleftarrow{\$} \mathsf{SPFSS.Gen}(1^\lambda, (\mathbf{A}_0^0)^{p^i} \boxplus \mathbf{A}_1^0, (\boldsymbol{s}_0)^{p^i} \otimes \boldsymbol{s}_1)$,
   $(K_0^{4i+1}, K_1^{4i+1}) \xleftarrow{\$} \mathsf{SPFSS.Gen}(1^\lambda, (\mathbf{A}_0^0)^{p^i} \boxplus \mathbf{A}_1^1, (\boldsymbol{s}_0)^{p^i} \otimes \boldsymbol{e}_1)$,
   $(K_0^{4i+2}, K_1^{4i+2}) \xleftarrow{\$} \mathsf{SPFSS.Gen}(1^\lambda, (\mathbf{A}_0^1)^{p^i} \boxplus \mathbf{A}_1^0, (\boldsymbol{e}_0)^{p^i} \otimes \boldsymbol{s}_1)$,
   $(K_0^{4i+3}, K_1^{4i+3}) \xleftarrow{\$} \mathsf{SPFSS.Gen}(1^\lambda, (\mathbf{A}_0^1)^{p^i} \boxplus \mathbf{A}_1^1, (\boldsymbol{e}_0)^{p^i} \otimes \boldsymbol{e}_1)$,
3. Sample FSS keys (referring to $A^{(s)}, B^{(s)}$) as follows:
   $(K_0^{4k}, K_1^{4k}) \xleftarrow{\$} \mathsf{SPFSS.Gen}(1^\lambda, \mathbf{A}_0^0, \boldsymbol{s}_0)$,
   $(K_0^{4k+1}, K_1^{4k+1}) \xleftarrow{\$} \mathsf{SPFSS.Gen}(1^\lambda, \mathbf{A}_0^1, \boldsymbol{e}_0)$,
   $(K_0^{4k+2}, K_1^{4k+2}) \xleftarrow{\$} \mathsf{SPFSS.Gen}(1^\lambda, \mathbf{A}_1^0, \boldsymbol{s}_1)$,
   $(K_0^{4k+3}, K_1^{4k+3}) \xleftarrow{\$} \mathsf{SPFSS.Gen}(1^\lambda, \mathbf{A}_1^1, \boldsymbol{e}_1)$,
4. For $\sigma \in \{0,1\}$, let $\mathbf{k}_\sigma = \Big((K_\sigma^i)_{i\in[0,4k+3]}\Big)$.
5. Output $(\mathbf{k}_0, \mathbf{k}_1)$.

Expand: On input $(\sigma, \mathbf{k}_\sigma)$:

1. Parse $\mathbf{k}_\sigma$ as $\Big((K_\sigma^i)_{i\in[0,4k)}, K_\sigma^{4k}, K_\sigma^{4k+1}, K_\sigma^{4k+2}, K_\sigma^{4k+3}\Big)$.
2. For each $i \in [0, 4k+3]$, compute $u_{\sigma,i} \leftarrow \mathsf{SPFSS.FullEval}(\sigma, K_\sigma^i)$, viewed as $\mathcal{R}_k$ elements.

3. For each $j \in [0,k)$, according to Eq.(13), compute

$$\mathbf{z}_\sigma^{(j)} := \phi_k\Big(\sum_{i=0}^{k-1} \xi^{p^{i+j}} \cdot \big(a^{p^i+1} u_{\sigma,4i} + a^{p^i} u_{\sigma,4i+1} + a u_{\sigma,4i+2} + u_{\sigma,4i+3}\big)\Big).$$

4. Compute $\mathbf{v}_\sigma := \phi_k\big(a \cdot u_{\sigma,4k+2} + u_{\sigma,4k+3}\big) \in \mathbb{F}_{p^k}^N$. For $s \in [0,N')$, $i \in [0,k)$, view $\mathbf{v}_\sigma[ks+i] \in \mathbb{F}_{p^k}$ as an element of $\mathbb{F}_p^k$, denoted by $\vec{\alpha}_{\sigma,i}^{(s)}$. It holds that $\vec{A}_i^{(s)} = \vec{\alpha}_{0,i}^{(s)} + \vec{\alpha}_{1,i}^{(s)}$, where $\vec{A}_i^{(s)}$ is the i-th column of $A^{(s)}$.
5. For each $j \in [0,k)$, compute $\mathbf{b}_\sigma^{(j)} := \mathsf{Tr}\big(\xi^{p^j} \cdot (a \cdot u_{\sigma,4k} + u_{\sigma,4k+1})\big) \in \mathbb{F}_p^N$. For $s \in [0,N')$, $i,j \in [0,k)$, denote $\mathbf{b}_\sigma^{(j)}[ks+i]$ by $\beta_{\sigma,i,j}^{(s)}$. It holds that $B_{i,j}^{(s)}$ of $B^{(s)}$ equals to $\beta_{0,i,j}^{(s)} + \beta_{1,i,j}^{(s)}$.
6. For $s \in [0,N')$, $i,j \in [0,k)$, $\sigma \in \{0,1\}$, view $\mathbf{z}_\sigma^{(j)}[ks+i] \in \mathbb{F}_{p^k}$ as an element of $\mathbb{F}_p^k$, denoted by $\vec{v}_{\sigma,i,j}^{(s)}$. Compute $\vec{\gamma}_{\sigma,j}^{(s)} := \sum_{i=0}^{k-1} \vec{v}_{\sigma,i,j}^{(s)}$. It holds that $\vec{C}_j^{(s)} = \vec{\gamma}_{0,j}^{(s)} + \vec{\gamma}_{1,j}^{(s)} = \sum_{i=0}^{k-1} \vec{A}_i^{(s)} B_{i,j}^{(s)}$, where $\vec{C}_j^{(s)}$ is the j-th column of $C^{(s)}$.
7. Output $(\alpha_{\sigma,i,j}^{(s)}, \beta_{\sigma,i,j}^{(s)}, \gamma_{\sigma,i,j}^{(s)}) \in \mathbb{F}_p^3$, for $i,j \in [0,k), s \in [0,N')$.

[a] Each $j \in [0,N)$ corresponds to a basis $\prod_{i=1}^n \mathtt{X}_i^{j_i}$ of $\mathcal{R}_k$, where $j_i \in [0,p^k-1)$. For simplicity, we abuse the pre-defined bijection φ_k.

Theorem 7.3. *Assume a secure* FSS *scheme* SPFSS *for sums of point functions and QA-SD($\mathcal{R}_k, t$) is hard. Then there exists a PCG construction that generates $k \times k$ matrix multiplication triples over $\mathbb{F}_p$. If the* SPFSS *is based on a* PRG : $\{0,1\}^\lambda \to \{0,1\}^{2\lambda+2}$ *via the* PRG*-based construction from [BGI16], for generating $N' = \lfloor (p^k-1)^n/k \rfloor$ matrix triples, we have that:*

- *Each party's seed has maximum size around:* $(4kt^2+4t)\big((\log N - \log t + 1)(\lambda+2) + \lambda + k\log p\big)$ *bits.*
- *The computation of* ***Expand*** *can be done with at most* $(2+\lceil k/\lambda \rceil)4(k+1)Nt$ PRG *operations and* $O(kN\log N)$ *operations in* $\mathbb{F}_{p^k}$.

7.3 PCG for Subfield OLE

In this section, we present an efficient PCG construction for subfield OLE. Recall that in subfield OLE, party P_0 holds $\mathbf{b} \in \mathbb{F}_p^N, \mathbf{z}_0 \in \mathbb{F}_{p^k}^N$, while party P_1 holds $\mathbf{x}, \mathbf{z}_1 \in \mathbb{F}_{p^k}^N$, such that $\mathbf{b} * \mathbf{x} = \mathbf{z}_0 + \mathbf{z}_1$. This directly implies an OT construction.

Our construction. Recall that in Section 7.2 we show how to construct a PCG for correlated subfield OLE, and the PCG construction based on Eq.(12) essentially admits a PCG for subfield OLE. However, the construction allows for kN correlated subfield OLE correlations, and it seems wasteful to use only N of them as subfield OLE correlations. More specifically, the construction has seed size of $O_\lambda(4kt^2(\log N + k))$ for N subfield OLE correlations. We propose an approach to significantly reducing the seed size, by a multiplicative factor of at most $O(k)$.

From now on, we focus on the case $\mathbb{F}_p = \mathbb{F}_2$. In general, our approach works for subfield OLE over any $\mathbb{F}_p$. To start with, we give notations used in this section as follows. Let $\mathcal{R}_2 = \mathbb{F}_4[\mathtt{X}_1, \ldots, \mathtt{X}_n]/(\mathtt{X}_1^3 - 1, \ldots, \mathtt{X}_n^3 - 1) \simeq \mathbb{F}_4^N$, and $\mathcal{R}_k = \mathbb{F}_{2^k}[\mathtt{X}_1, \ldots, \mathtt{X}_n]/(\mathtt{X}_1^3 - 1, \ldots, \mathtt{X}_n^3 - 1) \simeq \mathbb{F}_{2^k}^N$, where $N = 3^n$. By Lemma 4.4, if k is even, there exists $\zeta \in \mathbb{F}_{2^k}$ s.t. $\mathbb{F}_{2^k} = \mathbb{F}_4(\zeta)$, and the isomorphism $\phi_2 : \mathcal{R}_2 \to \mathbb{F}_4^N$ actually determines an isomorphism $\phi_k : \mathcal{R}_k \to \mathbb{F}_{2^k}^N = \mathbb{F}_4^N(\zeta)$, by fixing a basis $(1, \zeta, \ldots, \zeta^{k/2-1})$. We will rely on two QA-SD assumptions, QA-SD$(\mathcal{R}_2, t)$ and QA-SD$(\mathcal{R}_k, t)$.

In a high level, the idea is to restrict a QA-SD sample $x_0 := a \cdot s_0 + e_0$ to be in $\mathcal{R}_2 \subset \mathcal{R}_k$, so that applying trace on x_0 would only lead to a constant number of cross-terms to be shared rather than $4k$ terms as in Eq.(12). Let $a_0 \xleftarrow{\$} \mathcal{R}_2$ and s_0, e_0 be random t-sparse elements of $\mathcal{R}_2$. And let $a_1 \xleftarrow{\$} \mathcal{R}_k$ and s_1, e_1 be random t-sparse elements of $\mathcal{R}_k$. We present a self-contained PCG construction for subfield OLE over $\mathbb{F}_2$ in $\mathsf{Cons}_{\mathsf{sOLE}}^{\mathbb{F}_2}$, which assumes both QA-SD$(\mathcal{R}_2, t)$, QA-SD$(\mathcal{R}_k, t)$ and employs SPFSS according to the following equation:

$$\begin{aligned} \mathsf{Tr}(x_0) \cdot x_1 &= (x_0^2 + x_0) \cdot x_1 \\ &= \big((a_0 s_0 + e_0)^2 + (a_0 s_0 + e_0)\big) \cdot (a_1 \cdot s_1 + e_1) \\ &= (a_0^2 s_0^2 + e_0^2 + a_0 s_0 + e_0) \cdot (a_1 \cdot s_1 + e_1). \end{aligned} \tag{14}$$

Construction 4: $\mathsf{Cons}_{\mathsf{sOLE}}^{\mathbb{F}_2}$

PARAMETER: Security parameter λ, noise weight $t = t(\lambda)$, $N = 3^n$, $\mathcal{R}_k = \mathbb{F}_{2^k}[\mathtt{X}_1, \ldots, \mathtt{X}_n]/(\mathtt{X}_1^3 - 1, \ldots, \mathtt{X}_n^3 - 1)$, $\mathcal{R}_2 = \mathbb{F}_4[\mathtt{X}_1, \ldots, \mathtt{X}_n]/(\mathtt{X}_1^3 - 1, \ldots, \mathtt{X}_n^3 - 1)$. Fix a natural embedding $\mathcal{R}_2 \hookrightarrow \mathcal{R}_k$. An FSS scheme $(\mathsf{SPFSS.Gen}, \mathsf{SPFSS.FullEval})$ for sums of t^2 point functions, with domain $[0, N)$[a] and range $\mathbb{F}_{2^k}$.
PUBLIC INPUT: A uniformly random $a \in \mathcal{R}_k$.
CORRELATION: After expansion, outputs $\mathbf{b} \in \mathbb{F}_2^N, \mathbf{z}_0 \in \mathbb{F}_{2^k}^N$ and $(\mathbf{x}, \mathbf{z}_1) \in \mathbb{F}_{2^k}^{2N}$ such that $\mathbf{b} * \mathbf{x} = \mathbf{z}_0 + \mathbf{z}_1$.
Gen: On input 1^λ:

1. For $\sigma \in \{0, 1\}$, sample random vectors $\mathbf{A}_\sigma^0, \mathbf{A}_\sigma^1 \xleftarrow{\$} [0, N)^t$. Sample $\boldsymbol{s}_0, \boldsymbol{e}_0 \leftarrow (\mathbb{F}_4^*)^t$, and $\boldsymbol{s}_1, \boldsymbol{e}_1 \xleftarrow{\$} (\mathbb{F}_{2^k}^*)^t$.
2. Sample FSS keys according to Eq.(14), namely as follows:
   $(K_0^1, K_1^1) \xleftarrow{\$} \mathsf{SPFSS.Gen}(1^\lambda, (\mathbf{A}_0^0)^2 \boxplus \mathbf{A}_1^0, (\boldsymbol{s}_0)^2 \otimes \boldsymbol{s}_1)$,
   $(K_0^2, K_1^2) \xleftarrow{\$} \mathsf{SPFSS.Gen}(1^\lambda, (\mathbf{A}_0^0)^2 \boxplus \mathbf{A}_1^1, (\boldsymbol{s}_0)^2 \otimes \boldsymbol{e}_1)$,
   $(K_0^3, K_1^3) \xleftarrow{\$} \mathsf{SPFSS.Gen}(1^\lambda, \mathbf{A}_0^0 \boxplus \mathbf{A}_1^0, \boldsymbol{s}_0 \otimes \boldsymbol{s}_1)$,
   $(K_0^4, K_1^4) \xleftarrow{\$} \mathsf{SPFSS.Gen}(1^\lambda, \mathbf{A}_0^0 \boxplus \mathbf{A}_1^1, \boldsymbol{s}_0 \otimes \boldsymbol{e}_1)$,
   $(K_0^5, K_1^5) \xleftarrow{\$} \mathsf{SPFSS.Gen}(1^\lambda, (\mathbf{A}_0^1)^2 \boxplus \mathbf{A}_1^0, (\boldsymbol{e}_0)^2 \otimes \boldsymbol{s}_1)$,
   $(K_0^6, K_1^6) \xleftarrow{\$} \mathsf{SPFSS.Gen}(1^\lambda, \mathbf{A}_0^1 \boxplus \mathbf{A}_1^0, \boldsymbol{e}_0 \otimes \boldsymbol{s}_1)$,
   $(K_0^7, K_1^7) \xleftarrow{\$} \mathsf{SPFSS.Gen}(1^\lambda, \mathbf{A}_0^1 \boxplus \mathbf{A}_1^1, \boldsymbol{e}_0 \otimes \boldsymbol{e}_1)$,
   $(K_0^8, K_1^8) \xleftarrow{\$} \mathsf{SPFSS.Gen}(1^\lambda, (\mathbf{A}_0^1)^2 \boxplus \mathbf{A}_1^1, (\boldsymbol{e}_0)^2 \otimes \boldsymbol{e}_1)$.

3. Output $(\mathbf{k}_0, \mathbf{k}_1)$, where $\mathbf{k}_0 = \Big((K_0^i)_{i\in[1,8]}, (\mathbf{A}_0^0, \boldsymbol{s}_0), (\mathbf{A}_0^1, \boldsymbol{e}_0)\Big)$, and $\mathbf{k}_1 = \Big((K_1^i)_{i\in[1,8]}, (\mathbf{A}_1^0, \boldsymbol{s}_1), (\mathbf{A}_1^1, \boldsymbol{e}_1)\Big)$.

Expand: On input $(\sigma, \mathbf{k}_\sigma)$:

1. Parse $\mathbf{k}_\sigma$ as $\Big((K_\sigma^i)_{i\in[1,8]}, (\mathbf{A}_\sigma^0, \boldsymbol{s}_\sigma), (\mathbf{A}_\sigma^1, \boldsymbol{e}_\sigma)\Big)$.
2. Define elements of $\mathcal{R}_k$:

$$s_\sigma = \sum_{l=1}^{t} \boldsymbol{s}_\sigma[l] \cdot \mathbf{A}_\sigma^0[l], \quad e_\sigma = \sum_{l=1}^{t} \boldsymbol{e}_\sigma[l] \cdot \mathbf{A}_\sigma^1[l].$$

   Note that actually $s_0, e_0 \in \mathcal{R}_2 \subset \mathcal{R}_k$.
3. If $\sigma = 0$, compute $\mathbf{b} = \mathsf{Tr}(a_0 \cdot s_0 + e_0)$; if $\sigma = 1$, compute $\mathbf{x} = \phi_k(a_1 \cdot s_1 + e_1)$.
4. For $i \in [1, 8]$, compute $u_{\sigma,i} \leftarrow \mathsf{SPFSS.FullEval}(\sigma, K_\sigma^i)$, viewed as $\mathcal{R}_k$ elements.
5. According to Eq.(14), compute

$$\mathbf{z}_\sigma := \phi_k(a_0^2 a_1 u_{\sigma,1} + a_0^2 u_{\sigma,2} + a_0 a_1 u_{\sigma,3} + a_0 u_{\sigma,4} + a_1 u_{\sigma,5} + a_1 u_{\sigma,6} + u_{\sigma,7} + u_{\sigma,8}).$$

6. Output $(\mathbf{b}, \mathbf{z}_0)$ and $(\mathbf{x}, \mathbf{z}_1)$.

[a] Each $j \in [0, N)$ corresponds to a basis $\vec{\mathtt{X}}^j := \prod_{i=1}^n \mathtt{X}_i^{j_i}$ of $\mathcal{R}_2 \hookrightarrow \mathcal{R}_k$, where $j_i \in [0, 3)$. For simplicity, we abuse the pre-defined bijection φ_2.

Theorem 7.4. *Assume a secure* FSS *scheme* SPFSS *for sums of point functions and both QA-SD($\mathcal{R}_2, t$), QA-SD($\mathcal{R}_k, t$) are hard. Then there exists a PCG construction that generates subfield OLE correlations for $\mathbb{F}_{2^k}$ over $\mathbb{F}_2$. If the* SPFSS *is based on a* $\mathsf{PRG} : \{0,1\}^\lambda \to \{0,1\}^{2\lambda+2}$ *via the* PRG*-based construction from [BGI16], for generating $N = 3^n$ subfield OLE correlations, we have that:*

- *Each party's seed has maximum size around:* $8t^2\big((\log N - \log t + 1)(\lambda + 2) + \lambda + k\big) + 2t(\log N + k)$ *bits.*
- *The computation of* ***Expand*** *can be done with at most* $(2 + \lceil k/\lambda \rceil)4Nt$ PRG *operations and* $O(N \log N)$ *operations in* $\mathbb{F}_{2^k}$.

Applications of Subfield OLE. A direct application of subfield OLE is that it implies string OT. Recall that $\mathsf{Cons}^{\mathbb{F}_2}_{\mathsf{sOLE}}$ generates $\mathbf{b} \in \mathbb{F}_2^N, \mathbf{z}_0 \in \mathbb{F}_{2^k}^N$ and $(\mathbf{x}, \mathbf{z}_1) \in \mathbb{F}_{2^k}^{2N}$ such that $\mathbf{b} * \mathbf{x} = \mathbf{z}_0 + \mathbf{z}_1$. Then for each $j \in [1, N]$, party P_0 can view $\mathbf{b}[j]$ as a bit, and $\mathbf{z}_0[j]$ as a k-bit string, while party P_1 views $\mathbf{x}[j], \mathbf{z}_1[j]$ as two k-bit strings. It holds that $\mathbf{z}_0[j] = \mathbf{z}_1[j]$ if $\mathbf{b}[j] = 0$, and $\mathbf{z}_0[j] = \mathbf{z}_1[j] \oplus \mathbf{x}[j]$ if $\mathbf{b}[j] = 1$, which is essentially a OT correlation where P_0 is the OT receiver with input bit $\mathbf{b}[j]$, output string $\mathbf{z}_0[j]$, and P_1 is the OT sender with input strings $\mathbf{z}_1[j], \mathbf{z}_1[j] \oplus \mathbf{x}[j]$.

To address the advantage of our string OT construction from subfield OLE, we briefly review the previous PCG construction [BCG+19a, BCG+19b] for string OT from subfield VOLE. Recall that in subfield VOLE, party P_0 holds

$\mathbf{b} \in \mathbb{F}_2^N, \mathbf{z}_0 \in \mathbb{F}_{2^k}^N$, while party P_1 holds $x \in \mathbb{F}_{2^k}, \mathbf{z}_1 \in \mathbb{F}_{2^k}^N$, such that $\mathbf{b} \cdot x = \mathbf{z}_0 + \mathbf{z}_1$. Then for each $j \in [1, N]$, $\mathrm{P}_0, \mathrm{P}_1$ holds $(\mathbf{b}[j], \mathbf{z}_0[j]), (\mathbf{z}_1[j], \mathbf{z}_1[j] \oplus x)$ with $\mathbf{z}_0[j] = \mathbf{z}_1[j] \oplus (\mathbf{b}[j] \cdot x)$, which are not random OT correlations, as P_1's N pairs of inputs $(\mathbf{z}_1[j], \mathbf{z}_1[j] \oplus x)$ share the same offset x. To obtain standard random OT, they further assume a correlation-robust hash function $H : \{0,1\}^k \to \{0,1\}^\lambda$, such that

$$\big(\mathbf{b}[j], H(\mathbf{z}_0[j])\big), \big(H(\mathbf{z}_1[j]), H(\mathbf{z}_1[j] \oplus x)\big)$$

looks independently and uniformly random.

As shown above, transforming subfield OLE into OT does not require any cryptographic assumption, since the offsets $\mathbf{x}[j]$ are independently random. Our PCG construction for subfield OLE only assumes QA-SD assumptions and a SPFSS scheme. To our best knowledge, no previous PCG construction generates OT correlations without assuming correlation-robust hash functions. As a downside, the seed size is slightly larger than [BCG+19a, BCG+19b].

8 PCG Setup Protocols from QA-SD

In this section, we show the distributed two-party setup protocols for PCGs for OLE correlations and authenticated multiplication triples assuming the pseudorandomness of QA-SD assumptions. For the sake of generality, we consider the case of $\mathbb{F}_p$. Taking $p = 2$ leads to OLEs over $\mathbb{F}_2$ and authenticated Boolean triples. Distributed setup protocols for our PCGs on other correlations (e.g., matrix triples) can be obtained in a similar way, thus omitted. Note that our PCG protocols can be used in a bootstrapping flavor, where a small part of the PCG outputs would be reserved for subsequent PCG seed generations. In the setup protocols, we focus on bootstrapping from a smaller size correlations. The initial correlations could be obtained via previous protocols [KOS16, KPR18, BCG+19b].

In Section 8.1, a protocol $\Pi^{\mathbb{F}_p}_{\mathsf{OLE-Setup}}$ securely realizes the functionality $\mathcal{F}^{\mathbb{F}_p}_{\mathsf{OLE-Setup}}$ that outputs the OLE seed against semi-honest adversaries. After receiving the OLE seed, the parties locally expand the seed to generate OLE correlations. Similar to [BCG+20], we present a protocol $\Pi^{\mathbb{F}_p}_{\mathsf{mal-OLE}}$ that securely realizes the corruptible OLE functionality $\mathcal{F}^{\mathbb{F}_p}_{\mathsf{mal-OLE}}$ in Section 8.2 against malicious adversaries. Then in Section 8.3, we present a protocol $\Pi^{\mathbb{F}_p}_{\mathsf{Auth-Triple}}$ that realizes the functionality $\mathcal{F}^{\mathbb{F}_p}_{\mathsf{Auth-Triple}}$ for authenticated multiplication triples with malicious security. Finally, in Section 8.4 we summarize the complexity for basic operations used in our setup protocols. The underlying functionalities e.g., $\mathcal{F}_{\mathsf{2PC}}$, $\mathcal{F}_{\mathsf{DPF}}$ are presented in Appendix C.

8.1 Semi-honest Distributed Setup from QA-SD

In this section, we show a protocol $\Pi^{\mathbb{F}_p}_{\mathsf{OLE-Setup}}$ that realizes the seed generation functionality $\mathcal{F}^{\mathbb{F}_p}_{\mathsf{OLE-Setup}}$ for OLE correlations from QA-SD assumptions against a semi-honest adversary. Note that after the setup protocol, each party obtains

succinct representations of the sparse error vector via (postion, value) pairs and succinct representations of the products of error vectors via DPFs.

Functionality 5: $\mathcal{F}_{\mathsf{OLE-Setup}}^{\mathbb{F}_p}$

Parameters: Security parameter 1^λ, $\mathsf{PCG}_{\mathsf{OLE}} := (\mathsf{PCG}_{\mathsf{OLE}}.\mathsf{Gen}, \mathsf{PCG}_{\mathsf{OLE}}.\mathsf{Expand})$ in line with Construction $\mathsf{Cons}_{\mathsf{OLE}}^{\mathbb{F}_p}$.
Functionality:

1. Sample $(\mathsf{k}_0, \mathsf{k}_1) \leftarrow \mathsf{PCG}_{\mathsf{OLE}}.\mathsf{Gen}(1^\lambda)$.
2. For $\sigma \in \{0,1\}$, output k_σ to P_σ.

Then we show a protocol realizes the setup functionality $\mathcal{F}_{\mathsf{OLE-Setup}}^{\mathbb{F}_p}$ against semi-honest adversaries.

Protocol 6: $\Pi_{\mathsf{OLE-Setup}}^{\mathbb{F}_p}$

Parameters: Security parameter 1^λ, length $N = (p^k-1)^n$, $\mathbb{F} = \mathbb{F}_{p^k} = \mathbb{F}_p(\xi)$ and $\mathcal{R} = \mathbb{F}_{p^k}[\mathtt{X}_1, \ldots, \mathtt{X}_n]/(\mathtt{X}_1^{p^k-1}-1, \ldots, \mathtt{X}_n^{p^k-1}-1)$. Let $\mathbb{G} = \mathbb{Z}_{p^k-1}^n$. Let DPF be a distributed point function $\mathsf{DPF} := (\mathsf{DPF}.\mathsf{Gen}, \mathsf{DPF}.\mathsf{Eval})$ with domain size N and range $\mathbb{F}_{p^k}$. Assume each variable is shared via additive sharing, for instance $[x] = (x_0, x_1)$. Moreover, assume access to functionalities $\mathcal{F}_{\mathsf{2PC}}$, $\mathcal{F}_{\mathsf{DPF}}$.
Protocol:

1. P_σ samples random vectors $\mathbf{A}_\sigma^0, \mathbf{A}_\sigma^1 \xleftarrow{\$} \mathbb{G}^t$ and $\mathbf{b}_\sigma^0, \mathbf{b}_\sigma^1 \xleftarrow{\$} (\mathbb{F}_{p^k}^*)^t$. We remark that the pair $(\mathbf{A}_\sigma^i, \mathbf{b}_\sigma^i)$ defines a t-sparse element in $\mathcal{R}$.
2. P_0 inputs the positions and the values. For $i \in [0,1], j \in [1,t]$,
$$\left[\mathbf{A}_0^i[j]\right]^{\mathbb{G}} \leftarrow \mathsf{Input}(P_0, \mathbf{A}_0^i[j]), \left[\mathbf{b}_0^i[j]\right]^{\mathbb{F}} \leftarrow \mathsf{Input}(P_0, \mathbf{b}_0^i[j]).$$
3. P_1 computes the positions and the values. For $i \in [0,1], j \in [1,t], \ell \in [0, k-1]$, compute $p^\ell \cdot \mathbf{A}_1^i[j]$ and $(\mathbf{b}_1^i[j])^{p^\ell}$ iteratively. Then, P_1 inputs them
$$\left[p^\ell \cdot \mathbf{A}_1^i[j]\right]^{\mathbb{G}} \leftarrow \mathsf{Input}(P_1, p^\ell \cdot \mathbf{A}_1^i[j]), \left[(\mathbf{b}_1^i[j])^{p^\ell}\right]^{\mathbb{F}} \leftarrow \mathsf{Input}(P_1, (\mathbf{b}_1^i[j])^{p^\ell}).$$
4. Compute the cross sums of positions and products of values. For each $\ell \in [0, k-1]$, $i, j \in [1,t]$,
$$\left[\alpha_0^{ij\ell}\right]^{\mathbb{G}} \leftarrow \mathsf{Add}(\left[\mathbf{A}_0^0[i]\right], \left[p^\ell \cdot \mathbf{A}_1^0[j]\right]), \left[\beta_0^{ij\ell}\right]^{\mathbb{F}} \leftarrow \mathsf{Mul}(\left[\mathbf{b}_0^0[i]\right], \left[(\mathbf{b}_1^0[j])^{p^\ell}\right])$$
$$\left[\alpha_1^{ij\ell}\right]^{\mathbb{G}} \leftarrow \mathsf{Add}(\left[\mathbf{A}_0^1[i]\right], \left[p^\ell \cdot \mathbf{A}_1^0[j]\right]), \left[\beta_1^{ij\ell}\right]^{\mathbb{F}} \leftarrow \mathsf{Mul}(\left[\mathbf{b}_0^1[i]\right], \left[(\mathbf{b}_1^0[j])^{p^\ell}\right])$$
$$\left[\alpha_2^{ij\ell}\right]^{\mathbb{G}} \leftarrow \mathsf{Add}(\left[\mathbf{A}_0^0[i]\right], \left[p^\ell \cdot \mathbf{A}_1^1[j]\right]), \left[\beta_2^{ij\ell}\right]^{\mathbb{F}} \leftarrow \mathsf{Mul}(\left[\mathbf{b}_0^0[i]\right], \left[(\mathbf{b}_1^1[j])^{p^\ell}\right])$$
$$\left[\alpha_3^{ij\ell}\right]^{\mathbb{G}} \leftarrow \mathsf{Add}(\left[\mathbf{A}_0^1[i]\right], \left[p^\ell \cdot \mathbf{A}_1^1[j]\right]), \left[\beta_3^{ij\ell}\right]^{\mathbb{F}} \leftarrow \mathsf{Mul}(\left[\mathbf{b}_0^1[i]\right], \left[(\mathbf{b}_1^1[j])^{p^\ell}\right])$$

5. Convert the position value over $\mathbb{G}$ to binary value over $\{0,1\}^{\lceil \log |\mathbb{G}| \rceil}$. For $\kappa \in [0,3], i,j \in [1,t], \ell \in [0,k-1]$,

$$\left[\alpha_\kappa^{ij\ell}\right]^{\{0,1\}^{\lceil \log |\mathbb{G}| \rceil}} \leftarrow \mathsf{ToBinary}(\left[\alpha_\kappa^{ij\ell}\right]^{\mathbb{G}})$$

6. Sample the FSS key shares via calling $\mathcal{F}_{\mathsf{DPF}}$ with binary domain $\{0,1\}^{\lceil \log |\mathbb{G}| \rceil}$ and range $\mathbb{F}_{p^k}$. For each $\kappa \in [0,3], i,j \in [1,t], \ell \in [0,k-1]$,

$$(K_0^{ij\ell\kappa}, K_1^{ij\ell\kappa}) \leftarrow \mathcal{F}_{\mathsf{DPF}}(\left[\alpha_k^{ij\ell}\right]^{\{0,1\}^{\lceil \log |\mathbb{G}| \rceil}}, \left[\beta_k^{ij\ell}\right]^{\mathbb{F}}).$$

7. P_σ outputs $\mathsf{k}_\sigma := \left(\left\{K_\sigma^{ij\ell\kappa}\right\}_{i,j\in[1,t],\kappa\in[0,3]}^{\ell\in[0,k-1]}, \left\{\mathbf{A}_\sigma^i, \mathbf{b}_\sigma^i\right\}_{i\in\{0,1\}}\right)$.

Theorem 8.1. *The protocol $\Pi_{\mathsf{OLE-Setup}}^{\mathbb{F}_p}$ securely realizes the PCG seed generation functionality $\mathcal{F}_{\mathsf{OLE-Setup}}^{\mathbb{F}_p}$ with security against semi-honest adversaries in the $(\mathcal{F}_{\mathsf{2PC}}, \mathcal{F}_{\mathsf{DPF}})$-hybrid model.*

Proof. Note that the protocol $\Pi_{\mathsf{OLE-Setup}}^{\mathbb{F}_p}$ securely evaluates each step of $\mathsf{PCG}_{\mathsf{OLE.Gen}}$ of $\mathsf{Cons}_{\mathsf{OLE}}^{\mathbb{F}_p}$. The $\mathsf{SPFSS.Gen}$ is implemented via calling the $\mathcal{F}_{\mathsf{DPF}}$ upon each nonzero point. □

Remark 8.1. The above $\Pi_{\mathsf{OLE-Setup}}^{\mathbb{F}_p}$ considers the ring $\mathcal{R} = \mathbb{F}_{p^k}[\mathtt{X}_1 \dots \mathtt{X}_n]/(\mathtt{X}_1^{p^k-1} - 1, \dots, \mathtt{X}_n^{p^k-1} - 1)$, and the "additions" of positions are essentially computed over $\mathbb{G} = \mathbb{Z}_{p^k-1}^n$. For $p^k = 4$, the DPF KeyGen can be implemented via a ternary tree evaluation which requires 1-out-of-3 OT [BBC+24, Section 5]. The early termination technique [BGI16, Remark 3.4] can be used to reduce the depth of the GGM tree. Generally speaking, the early termination technique does not work for malicious DPF key generation. The main reason comes from the fact that, to achieve malicious security, DPF is used to succinctly share $K \cdot \mathbf{e}$, where K is a λ-bit global MAC key. In addition, the half-tree technique [GYW+23] can be used to further optimize the efficiency of underlying DPFs.

The costs of above setup protocol mainly consist of the following operations.

1. kt^2 Mul over $\mathbb{F}_{p^k}$.
2. $4kt^2$ $\mathsf{ToBinary}$ function calls over $\mathbb{Z}_{p^k-1}^n$.
3. $4kt^2$ $\mathcal{F}_{\mathsf{DPF}}$ calls with domain size N and range $\mathbb{F}_{p^k}$.

Combining with complexity for $\mathsf{ToBinary}$ and $\mathcal{F}_{\mathsf{DPF}}$, we have

Theorem 8.2 (Semi-honest Distributed OLE Generation). *Assume the hardness of QA-SD$(\mathcal{R}, t)$. Then there exists a protocol securely realizing $\mathcal{F}_{\mathsf{OLE-Setup}}$ against semi-honest adversaries. To generate kN OLE correlations over $\mathbb{F}_p$, the protocol has the following complexity.*

– ***Correlated randomness***: *Taking correlated randomness as follows.*
  1. kt^2 *multiplication triples over* $\mathbb{F}_{p^k}$.
  2. *Length* $4kt^2$ *subfield VOLE over* $\mathbb{F}_2/\mathbb{F}_{2^\lambda}$.
  3. $8kt^2$ *multiplication triples over* $\mathbb{F}_{p^k}$ *and* $4kt^2 \log N$ *number of* λ*-string-OTs.*
– ***Computational complexity***: *Dominated by* $8kt^2N$ *PRG calls.*
– ***Communication complexity***: *Dominated by*
  1. $2k^2t^2 \log p$ *bits.*
  2. $4k^2t^2n(\lambda+1)\log p$.
  3. $4kt^2((2\lambda+3)\log N + 5k \log p)$ *bits.*

8.2 OLE Setup Protocols from QA-SD with Malicious Security

We show PCG protocol for generating OLE correlations with distributed setup realizing the corruptible OLE functionality $\mathcal{F}^{\mathbb{F}_p}_{\mathsf{mal-OLE}}$ with security against malicious adversaries. The protocols with malicious security are based on a static leakage variant of QA-SD as in [BCG+20, Section 6.2]. Intuitively, in the distributed setup protocol, an adversary is able to maliciously guess a predicate on the position of the error vector.

Definition 8.1 (QA-SD with Static Leakage). *Let* $\mathcal{R} = \mathbb{F}_{p^k}[\mathbb{G}]$ *be a group algebra with* $|\mathbb{G}| = N$[6]. *Let* $\mathcal{S}_t$ *be the distribution over* $\mathcal{R}$ *with Hamming weight at most* t. *Let* $\mathsf{Game}(b, \lambda)$ *be the following game.*

1. *Assume* $\mathbf{A}^i \xleftarrow{\$} \mathbb{G}^t$ *and* $\mathbf{b}^i \xleftarrow{\$} (\mathbb{F}^*_{p^k})^t$
2. *The adversary* $\mathcal{A}$ *makes a polynomial number of queries of the form* (i, j, P) *where* $i \in \{0,1\}, j \in [1,t]$ *and* $P : [0,N) \to \{0,1\}$. *If a query satisfy* $P(\mathbf{A}^i[j]) = 0$, *then abort. Otherwise send* $\mathsf{continue}$ *to* $\mathcal{A}$.
3. *Set* $\mathbf{e}_i = \sum_j \mathbf{b}^i[j] \cdot \mathcal{A}^i[j]$ *and* $\mathbf{b}_0 \leftarrow \mathbf{a} \cdot \mathbf{e}_0 + \mathbf{e}_1$ *and* $\mathbf{b}_1 \xleftarrow{\$} \mathcal{R}$.
4. *Return* $\mathbf{b}_b$ *to* $\mathcal{A}$.

The QA-SD problem with static leakage for a given $\mathbf{a} \xleftarrow{\$} \mathcal{R}$ *is hard for an arbitrary* PPT *adversary* $\mathcal{A}$ *if*

$$|\Pr[\mathcal{A}(\mathsf{Game}(0,\lambda)) = 1] - \Pr[\mathcal{A}(\mathsf{Game}(1,\lambda)) = 1]| \leq \mathsf{negl}(\lambda).$$

Our protocol for OLE is given in $\Pi^{\mathbb{F}_p}_{\mathsf{mal-OLE}}$, which realizes the corruptible OLE functionality $\mathcal{F}^{\mathbb{F}_p}_{\mathsf{mal-OLE}}$ with malicious security.

[6] By instantiating $\mathbb{G}$ with $\prod_{i=1}^n (\mathbb{Z}/d_i\mathbb{Z})$, the group algebra $\mathbb{F}_{p^k}[\mathbb{G}]$ is nothing else than the ring $\mathbb{F}_{p^k}[\mathtt{X}_1, \ldots, \mathtt{X}_n]/(\mathtt{X}_1^{d_1} - 1, \ldots, \mathtt{X}_n^{d_n} - 1)$, where $d_i = p^k - 1$.

Functionality 7: $\mathcal{F}_{\mathsf{mal-OLE}}^{\mathbb{F}_p}$

Parameters: Security parameter 1^λ. Length parameter $N = (p^k - 1)^n$.
Functionality:
If both parties are honest:

1. Sample $\mathbf{x}_0, \mathbf{x}_1 \xleftarrow{\$} \mathbb{F}_p^{kN}$.
2. Sample $\mathbf{z}_0 \xleftarrow{\$} \mathbb{F}_p^{kN}$ and set $\mathbf{z}_1 = \mathbf{x}_0 * \mathbf{x}_1 - \mathbf{z}_0$.

If P_σ is corrupted,

1. Wait for input $(\mathbf{x}_\sigma, \mathbf{z}_\sigma) \in \mathbb{F}_p^{2kN}$ from the adversary.
2. Sample $\mathbf{x}_{1-\sigma}$ and compute $\mathbf{z}_{1-\sigma} \leftarrow \mathbf{x}_0 * \mathbf{x}_1 - \mathbf{z}_\sigma$.
3. Output $(\mathbf{x}_{1-\sigma}, \mathbf{z}_{1-\sigma})$ to the honest party $P_{1-\sigma}$.

Protocol 8: $\Pi_{\mathsf{mal-OLE}}^{\mathbb{F}_p}$

Parameters:Security parameter 1^λ, noise weight $t = t(\lambda)$, length $N = (p^k-1)^n$, $\mathbb{F} := \mathbb{F}_{p^k} = \mathbb{F}_p(\xi)$ and $\mathcal{R}_k = \mathbb{F}_{p^k}[\mathtt{X}_1, \ldots, \mathtt{X}_n]/(\mathtt{X}_1^{p^k-1} - 1, \ldots, \mathtt{X}_n^{p^k-1} - 1)$. Let $\eta \in \mathbb{N}$ such that $p^\eta \geq 2^\lambda$ and $k \mid \eta$. Let $\mathbb{G} = \mathbb{Z}_{p^k-1}^n$. Let DPF be a distributed point function $\mathsf{DPF} := (\mathsf{DPF.Gen}, \mathsf{DPF.Eval})$ with domain size N and range $\mathbb{F}_{p^k}$. We abuse ϕ as a fixed isomorphism map from $\mathcal{R}_k$ to $\mathbb{F}_{p^k}^N$, and from $\mathcal{R}_\eta$ to $\mathbb{F}_\eta^N$. Assume each variable is shared via a SPDZ-style authenticated sharing, for instance $[\![x]\!] := ([\![x]\!]_0, [\![x]\!]_1)$ and $[\![x]\!]_\sigma := (x_\sigma, M_\sigma[x])$ where $x = x_0 + x_1$, $M_0[x] + M_1[x] = x \cdot (\Delta_0 + \Delta_1)$ and (Δ_0, Δ_1) is a sharing of the global key. For simplicity, the malicious PowerP operation is implicitly represented as $[\![\cdot]\!]^{p^\ell}$. There exists an implicit $\mathsf{ToBinary}$ function call before calling $\mathcal{F}_{\mathsf{mal-DPF}}$. Furthermore, it is given access to the functionalities $\mathcal{F}_{\mathsf{2PC}}$ and $\mathcal{F}_{\mathsf{mal-DPF}}$.
Input: A random element $a \in \mathcal{R}_k$.
Correlation: For $\theta \in [0, k-1]$, output $\mathbf{X}_0^{(\theta)}, \mathbf{Z}_0^{(\theta)}, \mathbf{X}_1^{(\theta)}, \mathbf{Z}_1^{(\theta)} \in \mathbb{F}_p^N$ such that $\mathbf{X}_0^{(\theta)} * \mathbf{X}_1^{(\theta)} = \mathbf{Z}_0^{(\theta)} + \mathbf{Z}_1^{(\theta)}$.
Protocol:

1. P_σ samples random vectors $\mathbf{A}_\sigma^0, \mathbf{A}_\sigma^1 \xleftarrow{\$} \mathbb{G}^t$ and $\mathbf{b}_\sigma^0, \mathbf{b}_\sigma^1 \xleftarrow{\$} (\mathbb{F}^*)^t$. Note that each pair $(\mathbf{A}_\sigma^i, \mathbf{b}_\sigma^i)$ defines a t-sparse element in $\mathcal{R}_k$.
2. Input the position and values. For $\sigma \in \{0,1\}$, $i \in [0,1]$, $j \in [1,t]$, $[\![\mathbf{A}_\sigma^i[j]]\!] \leftarrow \mathsf{Input}(P_\sigma, \mathbf{A}_\sigma^i[j])$, $[\![\mathbf{b}_\sigma^i[j]]\!] \leftarrow \mathsf{Input}(P_\sigma, \mathbf{b}_\sigma^i[j])$
3. Generate FSS keys for $\left[\!\left[\mathbf{Z}^{(\theta)}\right]\!\right]$ according to Eq. (10). For $i, j \in [1,t]$, $\kappa, \ell \in [0, k-1]$, //Note there exists an implicit $\mathsf{ToBinary}$ function call before calling $\mathcal{F}_{\mathsf{mal-DPF}}$.

$$(K_0^{ij\kappa\ell 0}, K_1^{ij\kappa\ell 0}) \leftarrow \mathcal{F}_{\mathsf{mal-DPF}}(p^\kappa [\![\mathbf{A}_0^0[i]]\!] + p^\ell [\![\mathbf{A}_1^0[j]]\!], [\![\mathbf{b}_0^0[i]]\!]^{p^\kappa} \cdot [\![\mathbf{b}_1^0[j]]\!]^{p^\ell})$$

$$(K_0^{ij\kappa\ell 1}, K_1^{ij\kappa\ell 1}) \leftarrow \mathcal{F}_{\mathsf{mal-DPF}}(p^\kappa [\![\mathbf{A}_0^0[i]]\!] + p^\ell [\![\mathbf{A}_1^1[j]]\!], [\![\mathbf{b}_0^0[i]]\!]^{p^\kappa} \cdot [\![\mathbf{b}_1^1[j]]\!]^{p^\ell})$$

$$(K_0^{ij\kappa\ell 2}, K_1^{ij\kappa\ell 2}) \leftarrow \mathcal{F}_{\mathsf{mal-DPF}}(p^\kappa [\![\mathbf{A}_0^1[i]]\!] + p^\ell [\![\mathbf{A}_1^0[j]]\!], [\![\mathbf{b}_0^1[i]]\!]^{p^\kappa} \cdot [\![\mathbf{b}_1^0[j]]\!]^{p^\ell})$$

$(K_0^{ij\kappa\ell 3}, K_1^{ij\kappa\ell 3}) \leftarrow \mathcal{F}_{\mathsf{mal-DPF}}(p^\kappa [\![\mathbf{A}_0^1[i]]\!] + p^\ell [\![\mathbf{A}_1^1[j]]\!], [\![\mathbf{b}_0^1[i]]\!]^{p^\kappa} \cdot [\![\mathbf{b}_1^1[j]]\!]^{p^\ell})$

4. Generate $\mathbf{X}_\sigma^{(\theta)}$. P_σ computes

$$e_\sigma^0 = \sum_{j\in[1,t]} \mathbf{b}_\sigma^0[j] \cdot \vec{\mathtt{X}}^{\mathbf{A}_\sigma^0[j]}, \quad e_\sigma^1 = \sum_{j\in[1,t]} \mathbf{b}_\sigma^1[j] \cdot \vec{\mathtt{X}}^{\mathbf{A}_\sigma^1[j]}$$

and

$$\mathbf{X}_\sigma^{(\theta)} = \mathsf{Tr}\big(\xi^{p^\theta} \cdot (a \cdot e_\sigma^0 + e_\sigma^1)\big).$$

5. Generate $[\![\mathbf{Z}^{(\theta)}]\!]$. Set

$$K_\sigma^{\kappa\ell 0} := \sum_{i,j\in[t]} K_\sigma^{ij\kappa\ell 0}, \quad K_\sigma^{\kappa\ell 1} := \sum_{i,j\in[t]} K_\sigma^{ij\kappa\ell 1},$$

$$K_\sigma^{\kappa\ell 2} := \sum_{i,j\in[t]} K_\sigma^{ij\kappa\ell 2}, \quad K_\sigma^{\kappa\ell 3} := \sum_{i,j\in[t]} K_\sigma^{ij\kappa\ell 3}.$$

For $\theta \in [0, k-1]$, compute

$$[\![\mathbf{Z}^{(\theta)}]\!]_\sigma := \phi\Big(\sum_{\kappa,\ell\in[0,k-1]} \xi^{p^\theta(p^\kappa+p^\ell)} \cdot \big(a^{p^\kappa+p^\ell} \cdot K_\sigma^{\kappa\ell 0} + a^{p^\kappa} \cdot K_\sigma^{\kappa\ell 1} + a^{p^\ell} \cdot K_\sigma^{\kappa\ell 2} + K_\sigma^{\kappa\ell 3}\big)\Big)$$

6. P_σ outputs $\big(\mathbf{X}_\sigma^{(\theta)}, \mathbf{Z}_\sigma^{(\theta)}\big)$. //Note that we only output the sharing of $\mathbf{Z}^{(\theta)}$ without the MAC shares, though we actually get $[\![\mathbf{Z}^{(\theta)}]\!]$.

Note that the PowerP operation, i.e., $[\![\cdot]\!]^{p^\kappa}$ during FSS key generation for $[\![\mathbf{Z}^{(\theta)}]\!]$ can be precomputed, which avoids $O(k^2t^2)$ number of PowerP operations because the values of PowerP are reused. The complexity of PowerP is shown in Section 8.4.

Theorem 8.3. *If the QA-SD problem with static leakage is hard, then the protocol $\Pi_{\mathsf{mal-OLE}}^{\mathbb{F}_p}$ implements the functionality $\mathcal{F}_{\mathsf{mal-OLE}}^{\mathbb{F}_p}$ in the $(\mathcal{F}_{\mathsf{2PC}}, \mathcal{F}_{\mathsf{mal-DPF}})$-hybrid model against malicious adversaries.*

Proof. – Both parties are honest. Note that there is no direct communication between the two parties and the communication is performed via ideal functionality calls. Thus, we only need to prove that protocol the output distribution is computationally indistinguishable from the distribution of the ideal functionality description. From the pseudorandomness of QA-SD and the linear independence of the trace function, $(\mathbf{X}_0^{(\theta)}, \mathbf{X}_1^{(\theta)}, \mathbf{Z}_0^{(\theta)}, \mathbf{Z}_1^{(\theta)})$ is a pseudorandom OLE correlation.

– Corrupted P_σ. The simulator $\mathcal{S}$ waits for the input $(\mathbf{A}_\sigma^i, \mathbf{b}_\sigma^i)$ from $\mathcal{A}$ and sets, $e_\sigma^0 = \sum_{j\in[1,t]} \mathbf{b}_\sigma^0[j] \cdot \mathtt{X}^{\mathbf{A}_\sigma^0[j]}, e_\sigma^1 = \sum_{j\in[1,t]} \mathbf{b}_\sigma^1[j] \cdot \mathtt{X}^{\mathbf{A}_\sigma^1[j]}$ and $\mathbf{X}_\sigma^{(\theta)} = \phi(\mathsf{Tr}(\xi^{p^\theta} \cdot (a \cdot e_\sigma^0 + e_\sigma^1)))$. Next, $\mathcal{S}$ simulates the invocations to $\mathcal{F}_{\mathsf{mal-DPF}}$. For

$i \in \{0,1\}, j \in [t]$, $\mathcal{S}$ outputs $\beta = 0$ and aborts on that instance. After receiving a guess set B of size at most N from $\mathcal{A}$ for other instances, $\mathcal{S}$ responds with $\perp$ whatever B is and continues. For $i, j \in [t]$, $\kappa, \ell \in [0, k-1], \eta \in [0,3]$ of non-aborting instance, $\mathcal{S}$ awaits the input $K_\sigma^{ij\kappa\ell\eta}$ and predicate $P^{ij\kappa\ell\eta}$ from $\mathcal{A}$. Then $\mathcal{S}$ chooses $\mathbf{A}^0, \mathbf{A}^1, \mathbf{b}^0, \mathbf{b}^1$. For $i, j \in [t]$, $\kappa, \ell \in [0, k-1], \eta \in [0,3]$, in the $(i, j, \kappa, \ell, \eta)$ call of $\mathcal{F}_{\mathsf{mal-DPF}}$, if $P^{ij\kappa\ell\eta}(p^\kappa \mathbf{A}^\alpha[i] + p^\ell \mathbf{A}_\sigma^\beta[j]) = 0$ with $(\alpha\beta)$ being binary representation of η, $\mathcal{S}$ aborts on all instances and outputs $(p^\kappa \mathbf{A}^\alpha[i] + p^\ell \mathbf{A}_\sigma^\beta[j], (\mathbf{b}^\alpha[i])^{p^\kappa} \cdot (\mathbf{b}_\sigma^\beta[j])^{p^\ell})$. Otherwise, $\mathcal{S}$ outputs $\mathsf{success}$ and continues. If $\mathcal{S}$ does not abort on any instance, $\mathcal{S}$ defines $K_\sigma^{\kappa\ell\eta} = \sum_{i,j[t]} K_\sigma^{ij\kappa\ell\eta}$ and computes

$$\left[\!\left[\mathbf{Z}^{(\theta)}\right]\!\right]_\sigma := \phi\Big(\sum_{\kappa,\ell\in[0,k-1]} \xi^{p^\theta (p^\kappa + p^\ell)} \cdot \big(a^{p^\kappa + p^\ell} \cdot K_\sigma^{\kappa\ell 0} + a^{p^\kappa} \cdot K_\sigma^{\kappa\ell 1} + a^{p^\ell} \cdot K_\sigma^{\kappa\ell 2} + K_\sigma^{\kappa\ell 3}\big)\Big)$$

for $\theta \in [0, k-1]$. Then $\mathcal{S}$ forwards $(\mathbf{X}_\sigma^{(\theta)}, \mathbf{Z}_\sigma^{(\theta)})_{\theta\in[0,k-1]}$ to the functionality. Based on the above simulation, we show that if there exists an adversary $\mathcal{A}$ distinguish the above simulated transcript from a real transcript, then there exists an adversary $\mathcal{B}$ for the QA-SD problem with static leakage. $\mathcal{B}$ executes the simulation until it receives the predicate $P^{ij\kappa\ell\eta}$ from $\mathcal{A}$ during $\mathcal{F}_{\mathsf{mal-DPF}}$. $\mathcal{B}$ defines the predicate $Q^{ij\kappa\ell\eta} : \mathbb{G}^t \to \{0,1\}$ as $Q^{ij\kappa\ell\eta}(A) = 0$ if and only if $P^{ij\kappa\ell\eta}(A + p^\ell \mathbf{A}_\sigma^{\eta[2]}[j]) = 0$. Then $\mathcal{S}$ sends $(i, \kappa, \eta[1])$ and the predicate $Q^{ij\kappa\ell\eta}$ to the QA-SD game. If the game aborts on one $(\tilde{i}, \tilde{j}, \tilde{\kappa}, \tilde{\ell}, \tilde{\eta})$, then $\mathcal{B}$ samples $\mathbf{A}^0, \mathbf{A}^1 \leftarrow_R \mathbb{G}^t$ conditioned on $Q^{\tilde{i}\tilde{j}\tilde{\kappa}\tilde{\ell}\tilde{\eta}}(p^{\tilde{\kappa}} \mathbf{A}^{\tilde{\eta}[1]}[\tilde{i}]) = 0$ and $Q^{ij\kappa\ell\eta}(p^\kappa \mathbf{A}^{\eta[1]}[i]) = 1$ for queried $(i, j, \kappa, \ell, \eta)$. Next, $\mathcal{B}$ samples $\mathbf{b}^0, \mathbf{b}^1 \leftarrow_R (\mathbb{F}^*)^t$ and returns $(p^{\tilde{\kappa}} \mathbf{A}^{\tilde{\eta}[1]}[\tilde{i}] + p^{\tilde{\ell}} \mathbf{A}_\sigma^{\tilde{\eta}[2]}[\tilde{j}], (\mathbf{b}^{\tilde{\eta}[1]}[\tilde{i}])^{p^{\tilde{\kappa}}} \cdot (\mathbf{b}_\sigma^{\tilde{\eta}[2]}[\tilde{j}])^{p^{\tilde{\ell}}})$ to $\mathcal{A}$ and aborts. To enable the occurrence of the conditional event, $\mathcal{B}$ runs in expected polynomial time. If the QA-SD game does not abort, $\mathcal{B}$ receives $v \in \mathcal{R}$ and sets $\mathbf{X}_{1-\sigma} := \phi(v)$. Next $\mathcal{B}$ computes $\mathsf{Tr}(x_0) \cdot \mathsf{Tr}(x_1)$. Then $\mathcal{B}$ outputs 0 if $\mathcal{A}$ outputs real protocol execution.

Next, we argue that $\mathcal{B}$ statistically simulates the protocol. For $b = 0$, i.e., a QA-SD instance, $\mathcal{B}$ simulates the real protocol execution execept with negligible probability. Note that the probability that the adversary correctly guesses the B to the functionality $\mathcal{F}_{\mathsf{mal-DPF}}$ is negligible. The output distribution simulated by $\mathcal{B}$ is indeed the real output distribution. The QA-SD game aborts if and only if $\mathcal{F}_{\mathsf{mal-DPF}}$ aborts at least during one call. Thus, $\mathcal{B}$ simulates the real protocol execution except with negligible probability. For $b = 1$, $\mathcal{B}$ faithfully simulates the simulation, because the QA-SD game aborts if and only if the simulation aborts in an ideal execution. □

The above setup protocol mainly consists of the following steps.

1. $2t$ Input over $\mathbb{G}$ and $\mathbb{F}_{p^k}$.
2. $2tk$ PowerP operations over $\mathbb{F}_{p^k}$ and $4k^2t^2$ multiplications over $\mathbb{F}_{p^k}$.
3. $4t^2k^2$ $\mathsf{ToBinary}$ function calls over $\mathbb{G}$.

4. $4t^2k^2$ $\mathcal{F}_{\mathsf{mal-DPF}}$ functionality calls with domain size N and range $\mathbb{F}_{p^k}$.

Combining with complexity for PowerP, ToBinary and $\mathcal{F}_{\mathsf{mal-DPF}}$, we have

Theorem 8.4 (Malicious Distributed OLE Generation). *Assume the hardness of* QA-SD$(\mathcal{R}, t)$ *with static leakage. Then there exists a protocol securely realizing* $\mathcal{F}_{\mathsf{mal-OLE}}^{\mathbb{F}_p}$ *against malicious adversaries. To generate* kN *OLE correlations over* $\mathbb{F}_p$*, the protocol has the following complexity.*

- ***Correlated randomness****: Taking correlated randomness as follows:*
  1. *2 length* $2tn$ *subfield VOLE over* $\mathbb{Z}_{p^k-1}$ *and 2 length* $2t$ *subfield VOLE over* $\mathbb{F}_{p^k}$.
  2. $2tk$ *sVOLE over* $\mathbb{F}_{p^k}$ *and* $4k^2t^2$ *authenticated multiplication triples over* $\mathbb{F}_{p^k}$.
  3. $12t^2k^2 \log N$ *symmetric subfield VOLE and* $4t^2k^2 \log N$ *authenticated Boolean multiplication triples.*
  4. $12t^2k^2$ *authenticated multiplication triples over* $\mathbb{F}_{p^k}$ *and 2 length* $16t^2k^2$ *sVOLE over* $\mathbb{F}_{p^k}$.
- ***Computational complexity****: Dominated by* $8t^2k^2N$ *PRG calls and* $O(k^3N \log N)$ *operations over* $\mathbb{F}_{p^k} \times \mathbb{F}_{p^\eta}$.
- ***Communication complexity****: Dominated by*
  1. $2tnk \log p + 2tk \log p$ *bits per party.*
  2. $2tk^2 \log p + 4k^3t^2 \log p$ *bits per party.*
  3. $8t^2k^2(\lambda + 1) \log N$ *bits per party.*
  4. $4t^2k^2((2\lambda + 3) \log N + 11k \log p)$ *bits.*

8.3 Authenticated Multiplication Triples from QA-SD

We define the ideal corruptible functionality for authenticated Boolean triples in $\mathcal{F}_{\mathsf{Auth-Triple}}^{\mathbb{F}_p}$. We slightly modify the original construction for authenticated multiplication triples (with seed size increased by a small factor), so that the distributed setup can be simpler. Our protocol is obtained by extending the malicious OLE construction $\Pi_{\mathsf{mal-OLE}}^{\mathbb{F}_p}$ and the PCG for authenticated triple construction $\mathsf{Cons}'^{\mathbb{F}_p}_{\mathsf{Auth-Triple}}$.

Functionality 9: $\mathcal{F}_{\mathsf{auth-triple}}^{\mathbb{F}_p}$

Parameters: Security parameter 1^λ, $d \mid p^k - 1$, $N = d^n$, $\mathbb{F}_{p^k} = \mathbb{F}_p(\xi)$. Let $\eta \in \mathbb{N}$ such that $p^\eta \geq 2^\lambda$ and $k \mid \eta$.
Functionality:
If both parties are honest,

1. Sample $\Delta_0, \Delta_1 \xleftarrow{\$} \mathbb{F}_{p^\eta}$ and let $\Delta \leftarrow \Delta_0 + \Delta_1$.
2. Sample $\mathbf{x}_0, \mathbf{x}_1, \mathbf{y}_0, \mathbf{y}_1 \xleftarrow{\$} \mathbb{F}_p^{kN}$ and let $\mathbf{x} = \mathbf{x}_0 + \mathbf{x}_1, \mathbf{y} = \mathbf{y}_0 + \mathbf{y}_1$.
3. Let $\mathbf{z} \leftarrow \mathbf{x} * \mathbf{y} \in \mathbb{F}_p^{kN}$.

4. Sample $\mathbf{m}_{x,0}, \mathbf{m}_{y,0}, \mathbf{m}_{z,0} \xleftarrow{\$} \mathbb{F}_{p^\eta}^{kN}$ and let $\mathbf{m}_{x,1} \leftarrow \Delta \cdot \mathbf{x} - \mathbf{m}_{x,0}, \mathbf{m}_{y,1} \leftarrow \Delta \cdot \mathbf{y} - \mathbf{m}_{y,0}, \mathbf{m}_{z,1} \leftarrow \Delta \cdot \mathbf{z} - \mathbf{m}_{z,0}$.
5. For $\sigma \in \{0,1\}$, output $(\Delta_\sigma, \mathbf{x}_\sigma, \mathbf{y}_\sigma, \mathbf{z}_\sigma, \mathbf{m}_{x,\sigma}, \mathbf{m}_{y,\sigma}, \mathbf{m}_{z,\sigma})$ to P_σ.

If P_σ is corrupted,

1. Wait for input $(\Delta_\sigma, \mathbf{x}_\sigma, \mathbf{y}_\sigma, \mathbf{z}_\sigma, \mathbf{m}_{x,\sigma}, \mathbf{m}_{y,\sigma}, \mathbf{m}_{z,\sigma}) \in \mathbb{F}_{p^\eta} \times \mathbb{F}_p^{3kN} \times \mathbb{F}_{p^\eta}^{3kN}$ from the adversary.
2. Sample $\Delta_{1-\sigma} \xleftarrow{\$} \mathbb{F}_{p^\eta}$ and $\mathbf{x}_{1-\sigma}, \mathbf{y}_{1-\sigma} \xleftarrow{\$} \mathbb{F}_p^{kN}$. Set $\Delta \leftarrow \Delta_0 + \Delta_1, \mathbf{x} \leftarrow \mathbf{x}_0 + \mathbf{x}_1, \mathbf{y} \leftarrow \mathbf{y}_0 + \mathbf{y}_1$ and $\mathbf{z} = \mathbf{x} * \mathbf{y}$. Let $\mathbf{m}_{x,1-\sigma} \leftarrow \Delta \cdot \mathbf{x} - \mathbf{m}_{x,\sigma}, \mathbf{m}_{y,1-\sigma} \leftarrow \Delta \cdot \mathbf{y} - \mathbf{m}_{y,\sigma}, \mathbf{m}_{z,1-\sigma} \leftarrow \Delta \cdot \mathbf{z} - \mathbf{m}_{z,\sigma}$.
3. Output $(\Delta_{1-\sigma}, \mathbf{x}_{1-\sigma}, \mathbf{y}_{1-\sigma}, \mathbf{z}_{1-\sigma}, \mathbf{m}_{x,1-\sigma}, \mathbf{m}_{y,1-\sigma}, \mathbf{m}_{z,1-\sigma})$ to the honest party $P_{1-\sigma}$.

Construction 10: $\mathsf{Cons}'^{\mathbb{F}_p}_{\mathsf{Auth-Triple}}$

Parameters: Security parameter 1^λ, noise weight $t = t(\lambda)$, $d \mid p^k - 1$, $N = d^n$, η satisfying $p^\eta \geq 2^\lambda$ and $k \mid \eta$, $\mathcal{R}_k = \mathbb{F}_{p^k}[\mathtt{X}_1, \ldots, \mathtt{X}_n]/(\mathtt{X}_1^d - 1, \ldots, \mathtt{X}_n^d - 1)$, $\xi \in \mathbb{F}_{p^k}$ s.t. $\mathbb{F}_{p^k} = \mathbb{F}_p(\xi)$. Let $\mathbb{G} := \mathbb{Z}_d^n$ and $\mathbb{F} := \mathbb{F}_{p^k}$. An FSS scheme $(\mathsf{SPFSS.Gen}, \mathsf{SPFSS.FullEval})$ for sums of point functions, with domain size N and range $\mathbb{F}_{p^k}$ or $\mathbb{F}_{p^\eta}$. We abuse ϕ as a fixed isomorphism map from $\mathcal{R}_k$ to $\mathbb{F}_{p^k}^N$, and from $\mathcal{R}_\eta$ to $\mathbb{F}_{p^\eta}^N$. The global MAC key shares $\Delta_0, \Delta_1 \in \mathbb{F}_{p^\eta}$ are implicitly provided by authenticated multiplication triples.
Public input: A uniformly random $a \in \mathcal{R}_k$.
Correlation: For $\ell \in [0, k-1]$, output $(\left[\!\left[\mathbf{X}^{(\ell)}\right]\!\right], \left[\!\left[\mathbf{Y}^{(\ell)}\right]\!\right], \left[\!\left[\mathbf{Z}^{(\ell)}\right]\!\right])$, where $\mathbf{X}^{(\ell)}, \mathbf{Y}^{(\ell)}, \mathbf{Z}^{(\ell)} \in \mathbb{F}_p^N$ such that $\mathbf{X}^{(\ell)} * \mathbf{Y}^{(\ell)} = \mathbf{Z}^{(\ell)}$ and MAC key shares $\Delta_0, \Delta_1 \in \mathbb{F}_{p^\eta}$.
Gen: On input 1^λ,

1. For $\sigma \in \{0,1\}$, $i \in \{0,1\}$, sample random vectors $\mathbf{A}_\sigma^{x,i}, \mathbf{A}_\sigma^{y,i} \xleftarrow{\$} \mathbb{G}^t$ and $\mathbf{b}_\sigma^{x,i}, \mathbf{b}_\sigma^{y,i} \xleftarrow{\$} (\mathbb{F}^*)^t$ for the positions and values. Let $\mathbf{A}^{x,i}, \mathbf{A}^{y,i} \in \mathbb{G}^{2t}$ and $\mathbf{b}^{x,i}, \mathbf{b}^{y,i} \in (\mathbb{F}^*)^{2t}$ be the union of the corresponding positions and values. Define elements of $\mathcal{R}_k$:

$$e^{x,i} := \sum_{j=1}^{2t} \mathbf{b}^{x,i}[j] \cdot \vec{\mathtt{X}}^{\mathbf{A}^{x,i}[j]}, x := a \cdot e^{x,0} + e^{x,1}$$

$$e^{y,i} := \sum_{j=1}^{2t} \mathbf{b}^{y,i}[j] \cdot \vec{\mathtt{X}}^{\mathbf{A}^{y,i}[j]}, y := a \cdot e^{y,0} + e^{y,1}$$

2. Sample FSS keys for $\left[\!\left[\mathbf{X}^{(\ell)}\right]\!\right], \left[\!\left[\mathbf{Y}^{(\ell)}\right]\!\right]$ according to Eq.(9). For $i \in \{0,1\}$, $j \in [0, k-1]$, //Here we use $[\![\cdot]\!]$ to highlight that the MAC value is computed together.
   - $(K_0^{x,i,j}, K_1^{x,i,j}) \leftarrow \mathsf{SPFSS.Gen}(1^\lambda, \boxplus^{p^j} \left[\!\left[\mathbf{A}^{x,i}\right]\!\right], \otimes^{p^j} \left[\!\left[\mathbf{b}^{x,i}\right]\!\right])$

– $(K_0^{y,i,j}, K_1^{y,i,j}) \leftarrow \mathsf{SPFSS.Gen}(1^\lambda, \boxplus^{p^j} [\![\mathbf{A}^{y,i}]\!], \otimes^{p^j} [\![\mathbf{b}^{y,i}]\!])$

3. Sample FSS keys for $\left[\!\left[\mathbf{Z}^{(\ell)}\right]\!\right]$ according to Eq. (10). For $i, j \in [0, k-1]$,

$$(K_0^{z,i,j,0}, K_1^{z,i,j,0}) \leftarrow \mathsf{SPFSS.Gen}(1^\lambda, [\![\mathbf{A}^{x,0}]\!]^{p^i} \boxplus [\![\mathbf{A}^{y,0}]\!]^{p^j}, [\![\mathbf{b}^{x,0}]\!]^{p^i} \otimes [\![\mathbf{b}^{y,0}]\!]^{p^j})$$

$$(K_0^{z,i,j,1}, K_1^{z,i,j,1}) \leftarrow \mathsf{SPFSS.Gen}(1^\lambda, [\![\mathbf{A}^{x,0}]\!]^{p^i} \boxplus [\![\mathbf{A}^{y,1}]\!]^{p^j}, [\![\mathbf{b}^{x,0}]\!]^{p^i} \otimes [\![\mathbf{b}^{y,1}]\!]^{p^j})$$

$$(K_0^{z,i,j,2}, K_1^{z,i,j,2}) \leftarrow \mathsf{SPFSS.Gen}(1^\lambda, [\![\mathbf{A}^{x,1}]\!]^{p^i} \boxplus [\![\mathbf{A}^{y,0}]\!]^{p^j}, [\![\mathbf{b}^{x,1}]\!]^{p^i} \otimes [\![\mathbf{b}^{y,0}]\!]^{p^j})$$

$$(K_0^{z,i,j,3}, K_1^{z,i,j,3}) \leftarrow \mathsf{SPFSS.Gen}(1^\lambda, [\![\mathbf{A}^{x,1}]\!]^{p^i} \boxplus [\![\mathbf{A}^{y,1}]\!]^{p^j}, [\![\mathbf{b}^{x,1}]\!]^{p^i} \otimes [\![\mathbf{b}^{y,1}]\!]^{p^j})$$

4. For $\sigma \in \{0,1\}$, let $\mathsf{k}_\sigma = \left(\left\{K_\sigma^{x,i,j}, K_\sigma^{y,i,j}\right\}_{j\in[0,k-1]}^{i\in\{0,1\}}, \left\{K_\sigma^{z,i,j,\kappa}\right\}_{i,j\in[0,k-1]}^{\kappa\in[0,3]}\right)$.
5. Output $(\mathsf{k}_0, \mathsf{k}_1)$.

Expand: On input $(\sigma, \mathsf{k}_\sigma)$:

1. Parse k_σ as $\left(\left\{K_\sigma^{x,i,j}, K_\sigma^{y,i,j}\right\}_{j\in[0,k-1]}^{i\in\{0,1\}}, \left\{K_\sigma^{z,i,j,\kappa}\right\}_{i,j\in[0,k-1]}^{\kappa\in[0,3]}\right)$.
2. For $i \in \{0,1\}$, $j \in [0, k-1]$, compute //View $u^{i,j}, v^{i,j}, w^{ij\kappa}$ as $\mathcal{R}_k$ elements.

$$[\![u^{i,j}]\!]_\sigma \leftarrow \mathsf{SPFSS.FullEval}(\sigma, K_\sigma^{x,i,j}), [\![v^{i,j}]\!]_\sigma \leftarrow \mathsf{SPFSS.FullEval}(\sigma, K_\sigma^{y,i,j})$$

3. For $i, j \in [0, k-1], \kappa \in [0,3]$, compute

$$[\![w^{ij\kappa}]\!]_\sigma \leftarrow \mathsf{SPFSS.FullEval}(\sigma, K_\sigma^{z,i,j,\kappa}).$$

4. For $\ell \in [0, k-1]$, compute

$$[\![X^{(\ell)}]\!]_\sigma \leftarrow \phi\left(\sum_{j\in[0,k-1]} \xi^{p^{\ell+j}} \cdot \left(a^{p^j} \cdot [\![u^{0,j}]\!]_\sigma + [\![u^{1,j}]\!]_\sigma\right)\right)$$

$$[\![Y^{(\ell)}]\!]_\sigma \leftarrow \phi\left(\sum_{j\in[0,k-1]} \xi^{p^{\ell+j}} \cdot \left(a^{p^j} \cdot [\![v^{0,j}]\!]_\sigma + [\![v^{1,j}]\!]_\sigma\right)\right)$$

5. For $\ell \in [0, k-1]$, according to Eq.(10), compute

$$[\![Z^{(\ell)}]\!]_\sigma \leftarrow \phi\Big(\sum_{i,j\in[0,k-1]} \xi^{p^\ell(p^i+p^j)} \cdot (a^{p^i+p^j} \cdot [\![w^{z,i,j,0}]\!]_\sigma + a^{p^i} \cdot [\![w^{z,i,j,1}]\!]_\sigma + a^{p^j} \cdot [\![w^{z,i,j,2}]\!]_\sigma + [\![w^{z,i,j,3}]\!]_\sigma)\Big).$$

6. For $\ell \in [0, k-1]$, output $\left([\![\mathbf{X}^{(\ell)}]\!]_\sigma, [\![\mathbf{Y}^{(\ell)}]\!]_\sigma, [\![\mathbf{Z}^{(\ell)}]\!]_\sigma\right)$ to P_σ, where $\mathbf{X}^{(\ell)}, \mathbf{Y}^{(\ell)}, \mathbf{Z}^{(\ell)} \in \mathbb{F}_p^N$.

As indicated in $\mathsf{Cons}'^{\mathbb{F}_p}_{\mathsf{Auth-Triple}}$, the protocol for authenticated multiplication triples from QA-SD assumptions calls the $\mathcal{F}_{\mathsf{mal-DPF}}$, which not only outputs the

shared value but also the shared MACs for the shared value. To enable distributed setup and build on the QA-SD assumption with weight $2t$, for each of $\mathbf{X}, \mathbf{Y}$, each party samples a QA-SD instance with error weight $2t$ and thus the total weight for $\mathbf{X}$ is $4t$.

Protocol 11: $\Pi^{\mathbb{F}_p}_{\mathsf{auth-triple}}$

Parameters: Security parameter 1^λ, noise weight $t = t(\lambda)$, $d \mid p^k - 1$, length $N = d^n$, $\mathbb{F} := \mathbb{F}_{p^k} = \mathbb{F}_p(\xi)$ and $\mathcal{R} = \mathbb{F}_{p^k}[\mathtt{X}_1, \ldots, \mathtt{X}_n]/(\mathtt{X}_1^d - 1, \ldots, \mathtt{X}_n^d - 1)$. Let $\mathbb{G} = \mathbb{Z}_d^n$. Let $\eta \in \mathbb{N}$ such that $p^\eta \geq 2^\lambda$ and $k \mid \eta$. Let DPF be a distributed point function $\mathsf{DPF} := (\mathsf{DPF.Gen}, \mathsf{DPF.Eval})$ with domain size N and range $\mathbb{F}_{p^k}$ or $\mathbb{F}_{p^\eta}$. We abuse ϕ as a fixed isomorphism map from $\mathcal{R}_k$ to $\mathbb{F}_{p^k}^N$, and from $\mathcal{R}_\eta$ to $\mathbb{F}_{p^\eta}^N$. Assume each variable is shared via a SPDZ-style authenticated sharing, for instance $[\![x]\!] := ([\![x]\!]_0, [\![x]\!]_1)$ and $[\![x]\!]_\sigma := (x_\sigma, M_\sigma[x])$ where $x = x_0 + x_1$, $M_0[x] + M_1[x] = x \cdot (\Delta_0 + \Delta_1)$ and (Δ_0, Δ_1) is a sharing of the global key. For simplicity, the malicious PowerP operation is implicitly represented as $[\![\cdot]\!]^{p^\ell}$. There exists an implicit $\mathsf{ToBinary}$ function call before calling $\mathcal{F}_{\mathsf{mal-DPF}}$. Furthermore, it is given access to the functionalities $\mathcal{F}_{\mathsf{2PC}}$ and $\mathcal{F}_{\mathsf{mal-DPF}}$.
Input: A random element $a \in \mathcal{R}$.
Protocol:

1. The parties sample the error vector and input them. For $i \in \{0,1\}$, P_σ samples $\mathbf{A}_\sigma^{x,i}, \mathbf{A}_\sigma^{y,i} \xleftarrow{\$} \mathbb{G}^t$ and $\mathbf{b}_\sigma^{x,i}, \mathbf{b}_\sigma^{y,i} \xleftarrow{\$} (\mathbb{F}^*)^t$. For $i \in \{0,1\}, j \in [1,t]$,
   (i) $\left[\!\left[\mathbf{A}_\sigma^{x,i}[j]\right]\!\right] \leftarrow \mathsf{Input}(\sigma, \mathbf{A}_\sigma^{x,i}[j])$, $\left[\!\left[\mathbf{b}_\sigma^{x,i}[j]\right]\!\right] \leftarrow \mathsf{Input}(\sigma, \mathbf{b}_\sigma^{x,i}[j])$.
   (ii) $\left[\!\left[\mathbf{A}_\sigma^{y,i}[j]\right]\!\right] \leftarrow \mathsf{Input}(\sigma, \mathbf{A}_\sigma^{y,i}[j])$, $\left[\!\left[\mathbf{b}_\sigma^{y,i}[j]\right]\!\right] \leftarrow \mathsf{Input}(\sigma, \mathbf{b}_\sigma^{y,i}[j])$.
   Then P_σ obtains $\left\{ [\![\mathbf{A}^{x,i}[j]]\!]_\sigma, [\![\mathbf{b}^{x,i}[j]]\!]_\sigma, [\![\mathbf{A}^{y,i}[j]]\!]_\sigma, [\![\mathbf{b}^{y,i}[j]]\!]_\sigma \right\}_{i \in \{0,1\}, j \in [1,2t]}$.
2. Generate FSS keys for $\left(\left[\!\left[\mathbf{X}^{(\theta)}\right]\!\right], \left[\!\left[\mathbf{Y}^{(\theta)}\right]\!\right]\right)$ according to Eq.(9). For $i \in \{0,1\}$, $j \in [1,2t]$, $\ell \in [0, k-1]$, // Note there exists an implicit $\mathsf{ToBinary}$ function call before calling $\mathcal{F}_{\mathsf{mal-DPF}}$.

$$(K_0^{x,i,j,\ell}, K_1^{x,i,j,\ell}) \leftarrow \mathcal{F}_{\mathsf{mal-DPF}}(p^\ell \cdot \left[\!\left[\mathbf{A}^{x,i}[j]\right]\!\right], \left[\!\left[\mathbf{b}^{x,i}[j]\right]\!\right]^{p^\ell})$$

$$(K_0^{y,i,j,\ell}, K_1^{y,i,j,\ell}) \leftarrow \mathcal{F}_{\mathsf{mal-DPF}}(p^\ell \cdot \left[\!\left[\mathbf{A}^{y,i}[j]\right]\!\right], \left[\!\left[\mathbf{b}^{y,i}[j]\right]\!\right]^{p^\ell})$$

3. Generate FSS keys for $\left[\!\left[\mathbf{Z}^{(\theta)}\right]\!\right]$ according to Eq. (10). For $i, j \in [1, 2t]$, $\kappa, \ell \in [0, k-1]$,//Note there exists an implicit $\mathsf{ToBinary}$ function call before calling $\mathcal{F}_{\mathsf{mal-DPF}}$.

$$(K_0^{ij\kappa\ell 0}, K_1^{ij\kappa\ell 0}) \leftarrow \mathcal{F}_{\mathsf{mal-DPF}}(p^\kappa \left[\!\left[\mathbf{A}^{x,0}[i]\right]\!\right] + p^\ell \left[\!\left[\mathbf{A}^{y,0}[j]\right]\!\right], \left[\!\left[\mathbf{b}^{x,0}[i]\right]\!\right]^{p^\kappa} \cdot \left[\!\left[\mathbf{b}^{y,0}[j]\right]\!\right]^{p^\ell})$$

$$(K_0^{ij\kappa\ell 1}, K_1^{ij\kappa\ell 1}) \leftarrow \mathcal{F}_{\mathsf{mal-DPF}}(p^\kappa \left[\!\left[\mathbf{A}^{x,0}[i]\right]\!\right] + p^\ell \left[\!\left[\mathbf{A}^{y,1}[j]\right]\!\right], \left[\!\left[\mathbf{b}^{x,0}[i]\right]\!\right]^{p^\kappa} \cdot \left[\!\left[\mathbf{b}^{y,1}[j]\right]\!\right]^{p^\ell})$$

$$(K_0^{ij\kappa\ell 2}, K_1^{ij\kappa\ell 2}) \leftarrow \mathcal{F}_{\mathsf{mal-DPF}}(p^\kappa \left[\!\left[\mathbf{A}^{x,1}[i]\right]\!\right] + p^\ell \left[\!\left[\mathbf{A}^{y,0}[j]\right]\!\right], \left[\!\left[\mathbf{b}^{x,1}[i]\right]\!\right]^{p^\kappa} \cdot \left[\!\left[\mathbf{b}^{y,0}[j]\right]\!\right]^{p^\ell})$$

$$(K_0^{ij\kappa\ell 3}, K_1^{ij\kappa\ell 3}) \leftarrow \mathcal{F}_{\mathsf{mal-DPF}}(p^\kappa \left[\!\left[\mathbf{A}^{x,1}[i]\right]\!\right] + p^\ell \left[\!\left[\mathbf{A}^{y,1}[j]\right]\!\right], \left[\!\left[\mathbf{b}^{x,1}[i]\right]\!\right]^{p^\kappa} \cdot \left[\!\left[\mathbf{b}^{y,1}[j]\right]\!\right]^{p^\ell})$$

4. Generate $(\llbracket \mathbf{X}^{(\theta)} \rrbracket, \llbracket \mathbf{Y}^{(\theta)} \rrbracket)$. Set

$$K_\sigma^{x,i,\ell} := \sum_{j\in[1,2t]} K_\sigma^{x,i,j,\ell}, K_\sigma^{y,i,\ell} := \sum_{j\in[1,2t]} K_\sigma^{y,i,j,\ell}.$$

For $\theta \in [0, k-1]$, compute

$$\llbracket \mathbf{X}^{(\theta)} \rrbracket_\sigma := \phi\Big(\sum_{\ell\in[0,k-1]} \xi^{p^{\theta+\ell}} \cdot \big(a^{p^\ell} \cdot K_\sigma^{x,0,\ell} + K_\sigma^{x,1,\ell}\big)\Big)$$

$$\llbracket \mathbf{Y}^{(\theta)} \rrbracket_\sigma := \phi\Big(\sum_{\ell\in[0,k-1]} \xi^{p^{\theta+\ell}} \cdot \big(a^{p^\ell} \cdot K_\sigma^{y,0,\ell} + K_\sigma^{y,1,\ell}\big)\Big)$$

5. Generate $\llbracket \mathbf{Z}^{(\theta)} \rrbracket$. Set

$$K_\sigma^{\kappa\ell 0} := \sum_{i,j\in[2t]} K_\sigma^{ij\kappa\ell 0}, \quad K_\sigma^{\kappa\ell 1} := \sum_{i,j\in[2t]} K_\sigma^{ij\kappa\ell 1},$$

$$K_\sigma^{\kappa\ell 2} := \sum_{i,j\in[2t]} K_\sigma^{ij\kappa\ell 2}, \quad K_\sigma^{\kappa\ell 3} := \sum_{i,j\in[2t]} K_\sigma^{ij\kappa\ell 3}.$$

For $\theta \in [0, k-1]$, compute

$$\llbracket \mathbf{Z}^{(\theta)} \rrbracket_\sigma := \phi\Big(\sum_{\kappa,\ell\in[0,k-1]} \xi^{p^\theta(p^\kappa+p^\ell)} \cdot \big(a^{p^\kappa+p^\ell} K_\sigma^{\kappa\ell 0} + a^{p^\kappa} K_\sigma^{\kappa\ell 1} + a^{p^\ell} K_\sigma^{\kappa\ell 2} + K_\sigma^{\kappa\ell 3}\big)\Big).$$

6. P_σ outputs $\Big(\llbracket \mathbf{X}^{(\theta)} \rrbracket_\sigma, \llbracket \mathbf{Y}^{(\theta)} \rrbracket_\sigma, \llbracket \mathbf{Z}^{(\theta)} \rrbracket_\sigma\Big)$, for each $\theta \in [0, k-1]$.

Note that the PowerP operation during FSS key generations for $(\llbracket \mathbf{X} \rrbracket, \llbracket \mathbf{Y} \rrbracket)$ can be reused during FSS key generations for $\llbracket \mathbf{Z} \rrbracket$. The ToBinary function call during FSS key generations for $\llbracket \mathbf{Z} \rrbracket$ can be avoided via BitAdd the position values of FSS key for $(\llbracket \mathbf{X} \rrbracket, \llbracket \mathbf{Y} \rrbracket)$.

Theorem 8.5. *If the QA-SD problem with static leakage is hard, then the protocol* $\Pi^{\mathbb{F}_p}_{\mathsf{Auth-Triple}}$ *implements the functionality* $\mathcal{F}^{\mathbb{F}_p}_{\mathsf{Auth-Triple}}$ *in the* $(\mathcal{F}_{\mathsf{2PC}}, \mathcal{F}_{\mathsf{mal-DPF}})$*-hybrid model.*

The proof is an analogue to the proof of Theorem 8.3. We omit it here.

The above setup protocol mainly consists of the following steps.

1. $4t$ Input over $\mathbb{G}$ and $\mathbb{F}_{p^k}$.
2. $8tk$ PowerP operations over $\mathbb{F}_{p^k}$.
3. $8tk$ ToBinary function calls over $\mathbb{G}$.
4. $8tk$ $\mathcal{F}_{\mathsf{mal-DPF}}$ functionality calls with domain size N and range $\mathbb{F}_{p^k}$.
5. $16k^2t^2$ Mul over $\mathbb{F}_{p^k}$.
6. $16k^2t^2$ BitAdd function calls over $\mathbb{F}_2^{\log|\mathbb{G}|}$.
7. $16k^2t^2$ $\mathcal{F}_{\mathsf{mal-DPF}}$ functionality calls with domain size N and range $\mathbb{F}_{p^k}$.

Combining with complexity for PowerP, ToBinary and $\mathcal{F}_{\mathsf{mal-DPF}}$, we have

Theorem 8.6 (Distributed Authenticated Multiplication Triples Generation). *Assume the hardness of* QA-SD$(\mathcal{R}, t)$ *with static leakage. Then there exists a protocol securely realizing* $\mathcal{F}_{\mathsf{Auth-Triple}}^{\mathbb{F}_p}$ *against malicious adversaries. To generate* kN *authenticated multiplication triples over* $\mathbb{F}_p$*, the protocol has the following complexity.*

- ***Correlated randomness****: Taking correlated randomness as follows:*
  1. *2 length* $4tn$ *subfield VOLE over* $\mathbb{Z}_{p^k-1}$ *and 2 length* $4t$ *subfield VOLE over* $\mathbb{F}_{p^k}$*.*
  2. *2 length* $8tk$ *subfield VOLE over* $\mathbb{F}_{p^k}$*.*
  3. $24tk \log N$ *symmetric subfield VOLE and* $8tk \log N$ *authenticated Boolean multiplication triples.*
  4. $24tk$ *authenticated multiplication triples and 2 length* $32tk$ *sVOLE over* $\mathbb{F}_{p^k}$*.*
  5. $16k^2t^2$ *authenticated multiplication triples over* $\mathbb{F}_{p^k}$*.*
  6. $16k^2t^2$ *authenticated Boolean multiplication triples.*
  7. $48k^2t^2$ *authenticated multiplication triples and 2 length* $64k^2t^2$ *sVOLE over* $\mathbb{F}_{p^k}$*.*
- ***Computational complexity****: Dominated by* $(32t^2k^2 + 16tk)N$ *PRG calls and* $O(k^3 N \log N)$ *operations overs* $\mathbb{F}_{p^k} \times \mathbb{F}_{p^\eta}$*.*
- ***Communication complexity****: Dominated by*
  1. $4tnk \log p + 4tk \log p$ *bits per party.*
  2. $8tk^2 \log p$ *bits per party.*
  3. $16tk(\lambda + 1) \log N$ *bits per party.*
  4. $8tk((2\lambda + 3) \log N + 11k \log p)$ *bits per party.*
  5. $32k^3t^2 \log p$ *bits per party.*
  6. $32k^2t^2$ *bits per party.*
  7. $16k^2t^2((2\lambda + 3) \log N + 11k \log p)$ *bits per party.*

8.4 Complexity for Basic Operations

We list the complexity of the main functions in our protocol.
Input(P_σ, x): For semi-honest security, there is no communication, i.e., $[x] := (x, 0)$. For malicious security, it consumes an authenticated random value and has communication cost of $\log q$ bits per party for $x \in \mathbb{F}_q$. The authenticated random value can be implemented via two symmetric subfield VOLE [BCG+19b]. Assume $p^k - 1 = \prod_{i=1}^{\ell} p_i^{d_i}$. To support authenticated sharing over $\mathbb{Z}_{p^k-1}$, we simply use authenticated sharing over $\{\mathbb{Z}_{p_i^{d_i}}\}_{i\in[\ell]}$ via Chinese remainder theorem. For $d_i > 1$, Galois ring [EXY22] can be used to provide authenticated sharing. Here, we omit the details.
Add: It is a local operation.
Scalar : It is a local operation.
PowerP : For semi-honest security, it is a local operation and there is no communication because for any $x_1, x_2 \in \mathbb{F}_{p^k}$, $(x_1+x_2)^{p^i} = x_1^{p^i} + x_2^{p^i}$. For malicious security,

the parties locally compute shares and then consume a random authenticated value over $\mathbb{F}_{p^k}$ to obtain an authenticated shares. The online communication is $k \log p$ bits per party. Each random authenticated value consumes two symmetric sVOLE instances over $\mathbb{F}_{p^k}$. To verify the PowerP operation is honestly executed, for any uniformly distributed $x \in \mathbb{F}_{p^k}$, the parties consecutively compute $[\![x^p]\!] \dots \left[\!\left[x^{p^{k-1}}\right]\!\right], \left[\!\left[x^{p^k}\right]\!\right]$ and then verify $\left[\!\left[x^{p^k}\right]\!\right] - [\![x]\!] = [\![0]\!]$ because $x^{p^k} = x$ and x is uniformly distributed. Note that this verification is not trivial because after each PowerP operation the value shares and the MAC shares are refreshed by a random authenticated value. The verification cost can be amortized for a batch of opening via checking a random linear combination of the MAC shares.
$\mathsf{Mul}([\![x]\!]^{\mathbb{F}_q}, [\![y]\!]^{\mathbb{F}_q})$: It takes one multiplication triple and has online communication complexity of $2 \log q$ bits per party.
$\mathsf{ToBinary}$: For semi-honest security, we use the $\mathsf{A2B}$ algorithm in ABY2.0 [PSSY21, Lemma C.5] to convert a number to binary representations, which requires $(4\lambda + 1) \log N$ bits communication for setup phase and $(\lambda + 1) \log N$ bits online communication, and takes one correlated OT on λ bit strings or one subfield VOLE over $\mathbb{F}_2/\mathbb{F}_{2^\lambda}$. For malicious security, we use the authenticated garbled circuits of [WRK17, Table 2] to evaluate size-optimized adder circuit [BP06, SZ13] with size $\log N$ and depth $\log N$ to obtain Yao sharing and then use $\mathsf{Y2B}$ to obtain authenticated binary sharings. The online communication is $(2\lambda + 2) \log N$ bits. The authenticated garbled circuit takes $2 \log N$ symmetric subfield VOLE over $\mathbb{Z}_{p^k-1}$, $\log N$ symmetric subfield VOLE over $\mathbb{F}_2$, and $\log N$ authenticated multiplication triples over $\mathbb{F}_2$.
$\mathsf{BitAdd}([\![x]\!]^{\mathbb{F}_2^\ell}, [\![y]\!]^{\mathbb{F}_2^\ell})$: It computes the sum of two integers in binary representations and takes ℓ AND gates, i.e., ℓ Boolean multiplication triples, leading to 2ℓ online communication bits per party.

9 Security Analysis and Parameter Selections

In this section, we analyze the security of QA-SD and Ring-LPN problems under various concrete attacks to choose appropriate parameters for a given security level. The general field $\mathbb{F}_{p^k}$ is taken into account when investigating these attacks.

In particular, an adversary is given a QA-SD/Ring-LPN instance $(a, b) \in \mathcal{R}^2$ with $a \xleftarrow{\$} \mathcal{R}$, and $b = a \cdot e_0 + e_1$, where e_0, e_1 are random elements of $\mathcal{R}$ of Hamming weight t. The goal is to either distinguish (a, b) with the uniform distribution or to recover e_0, e_1. Essentially, the QA-SD/Ring-LPN correspond to linear codes over $\mathbb{F}_{p^k}$ of dimension N, length $2N$ and thus code rate $1/2$. Alternatively, the problems can be viewed as syndrome decoding problems, in the sense that given $(H, \mathbf{b} = H \cdot \mathbf{e})$ the adversary is asked to recover $\mathbf{e}$, where $H \in \mathbb{F}_{p^k}^{N \times 2N}$ is the parity check matrix and derived from a.

We consider two typical noise distributions:

Fixed Hamming weight. Let $\mathsf{HW}(\mathbb{F}) = \{\mathsf{HW}_{t,N}(\mathbb{F})\}_{t,N}$ be the family of distributions of uniformly random vectors with fixed Hamming weight. Namely, we abuse $\mathbf{e} \leftarrow \mathsf{HW}_{t,N}(\mathbb{F})$ as $\mathbf{e} \xleftarrow{\$} \{\mathbf{x} \in \mathbb{F}^N \mid \mathsf{wt}(\mathbf{x}) = t\}$.

Regular Hamming weight. Let $\mathsf{RG}(\mathbb{F}) = \{\mathsf{RG}_{t,N}(\mathbb{F})\}_{t,N}$ be the family of distributions of uniformly random regular weight vectors. W.o.l.g. we assume $t|N$ and let $\mathbf{e} = (\mathbf{e}^{(1)}, ..., \mathbf{e}^{(t)})$, where $\mathbf{e}^{(i)} \in \mathbb{F}^{N/t}, i \in [t]$. Similarly, we abuse $\mathbf{e} \leftarrow \mathsf{RG}_{t,N}(\mathbb{F})$ as $\mathbf{e} \xleftarrow{\$} \{\mathbf{x} \in \mathbb{F}^N \mid \mathsf{wt}(\mathbf{x}^{(i)}) = 1, i \in [t]\}$. Essentially, $\mathsf{RG}(\mathbb{F})$ can be viewed as a special case of $\mathsf{HW}(\mathbb{F})$.

In this section, we mainly focus on attacks for the fixed Hamming weight distributions. Besides, to our best knowledge, assuming the regular structure would not significantly reduce the complexity of applying these attacks on our concrete instantiation. Our analyses on QA-SD and Ring-LPN follow the routine of previous works [BCG+20, BCCD23, BBC+24].

In summary, our analyses show that first the BKW attacks do not work, since the code rate here is only 1/2, which can only apply to codes of extremely low code rates. Secondly, it suffices to consider the information set decoding (ISD) attacks and the statistical decoding (SD) attacks, whose complexity are exponential in the Hamming weight of the secret vector (e_0, e_1). Finally, there exists a folding attack that allows to fold a given instance into a new instance with reduced code dimension, code length, and almost preserved noise weight, so that one can apply a subsequent ISD attack. To avoid such folding attacks, for Ring-LPN, we can choose N to be a prime. However, it is unavoidable in the QA-SD setting. Hence, we take it into account.

Specifically, we select parameters for the particularly interesting Boolean case, i.e. OLE over $\mathbb{F}_2$ and authenticated Boolean triples either from QA-SD or Ring-LPN. In general, QA-SD assumptions offer more flexibility than Ring-LPN on the parameter choice.

9.1 Known Attacks for Ring-LPN and QA-SD

Gaussian Elimination Attacks on Syndrome Decoding. Given $H \in \mathbb{F}_q^{k \times n}, \mathbf{b} \in \mathbb{F}_q^n$, the goal of the syndrome decoding is to find $\mathbf{e} \in \mathbb{F}_q^n$ such that $H \cdot \mathbf{e} = \mathbf{b}$ and $\mathsf{wt}(\mathbf{e}) = w$. A direct Gaussian elimination attempts to guess the $n-k$ error-free positions and then run the Gaussian elimination to recover the secret vector. Averagely, the Gaussian elimination needs $1/(1-w/n)^{n-k}$ iterations to find one $n-k$ error free positions and during each iteration the attack inverts a $(n-k)\times(n-k)$ sub-matrix with complexity $O((n-k)^{2.8})$ by Straseen's technique. The quasi-abelian/quasi-cyclic structure enables faster inversion algorithm with complexity $O((n-k)^2 \log(n-k))$. Thus the cost becomes

$$O\left(\left(\frac{1}{1-w/n}\right)^{n-k} \cdot (n-k)^2 \log(n-k)\right) \approx O(e^{\frac{w}{n/(n-k)}} \cdot (n-k)^2 \log(n-k)).$$

Note that the regular noise structure would not decrease the costs of the attack. Assume the error vector is split into w sub-vectors of length n/w, each having one error entry. Then the attack finds $(n-k)/w$ error-free positions in each of the w sub-vectors. The success probability is

$$\left(\left(1-\frac{1}{n/w}\right)^{(n-k)/w}\right)^w = \left(1-\frac{w}{n}\right)^{n-k}.$$

Information Set Decoding Attacks on Syndrome Decoding. Generally speaking, the information set decoding (ISD) algorithm of Prange [Pra62] attempts to find a size w subset of the rows of H such that the w rows exactly span $\mathbf{b}$. The ISD algorithm has been studies in a long line of works [Ste88, FS09, BLP11, MMT11, BJMM12, MO15, EKM17, BM18, DEEK24]. For syndrome decoding problem with constant rate, and error weight $w << n$, the analysis of Torres and Sendrier [TS16] indicates that all known ISD variants have the following asymptotic complexity $c^{w(1+o(1))}$, where c is constant related to the code rate. As it is the case of our settings, the total complexity of applying Prange's ISD algorithm is

$$O\left(c^{w\cdot(1+o(1))}\cdot(n-k)^2\log(n-k)\right).$$

Statistical Decoding Attacks on Syndrome Decoding. In contrast to the previous attacks recovering $\mathbf{e}$, the statistical decoding(SD) attacks directly try to distinguish a Ring-LPN or QA-SD instance from a random instance. By the Singleton bound, the minimum distance of the code generated by H is lower bounded by $n-k+1$. Given $(H, \mathbf{b} = H\cdot\mathbf{e})$, the SD attacks search for $\mathbf{m}$ such that $\mathbf{m}^T\cdot H$ is of small weight and then test whether $\langle\mathbf{m},\mathbf{b}\rangle$ equals 0. If $\mathbf{b}$ is random, then $\langle\mathbf{m},\mathbf{b}\rangle$ being 0 happens with probability $1/q$. Otherwise, the probability of $\langle\mathbf{m},\mathbf{b}\rangle$ being 0 is $1/q+((k-1)/n)^w$. Thus, the distinguishing advantage is $((k-1)/n)^w$. Assume there exists a pre-processing procedure computing $\mathbf{m}$ such that $\mathbf{m}\cdot H$ is of small Hamming weight. The complexity of SD attack is lower bounded by

$$O\left(\left(\frac{n}{k-1}\right)^w\cdot k\right).$$

We remark again that it seems that the adversary is not able to utilize the regular noise structure to improve the cost of the SD attacks.

Attacks On LPN with Many Samples. Given a large number of LPN samples, there exist attacks with various time-space trade-offs. However, such attacks only work for LPN with at least super-linear number of samples. In our Ring-LPN and QA-SD assumptions, the number of samples is linear, i.e., $2N$ with respect to dimension N samples, rather than super-linear.

The well-known BKW [BKW00] algorithm has both sub-exponential sample complexity and sub-exponential time complexity, namely, $2^{O(N/\log N)}$. It is worth to mention that BKW can be viewed as a variant of the ISD attack. Note that BKW works for constant error rate as well. There exists an improved BKW algorithm [Lyu05] which has polynomial sample complexity, i.e., $\Omega(N^{1+\epsilon})$ and degenerated time complexity $2^{O(N/\log\log N)}$. Additionally, BKW can also be combined with other existing attacks. The resulting algorithm still has sub-exponential sample complexity and sub-exponential time complexity [EKM17].

Attacks on Ring-LPN with Fully Splittable Polynomials. Recall that for Ring-LPN assumptions, we use the ring $\mathcal{R} = \mathbb{F}_{p^k}[\mathtt{X}]/(\mathtt{X}^N-1)$, where $N \mid p^k-1$ so that $f(\mathtt{X}) := \mathtt{X}^N-1$ is fully splittable over $\mathbb{F}_{p^k}$. This will lead to improved attacks. Assume f has a sparse factor g, then the original instance can be reduced to a

new instance with reduced dimension and non-increasing noise weight. Then, the adversary can run the above mentioned attacks on the above reduced instance. The attack would be more efficient, since it has lower dimension.

Note that in our setting, $f(\mathtt{X}) = \mathtt{X}^N - 1$ completely splits into N linear factors over $\mathbb{F}_{p^k}$. For $n \mid N$, we have $f(\mathtt{X}) = (\mathtt{X}^n - 1) \cdot f'(\mathtt{X}) = (\mathtt{X}^{N/n} - 1) \cdot f''(\mathtt{X})$ where f', f'' are of degree $N - n, N - N/n$, respectively and maybe not sparse. Furthermore for even N, we have $f(\mathtt{X}) = (\mathtt{X}^{N/2} + 1)(\mathtt{X}^{N/2} - 1)$. For a prime N, any nontrivial degree-n factor $g(\mathtt{X})$ of $\mathtt{X}^N - 1$ has sparsity $n + 1$. We remark that if reducing by an n-sparse factor, typically one gets a factor of n amplification for the noise weight. For $n > 2$, this significantly increases the noise weight and the attack becomes infeasible. Thus, we focus on the 2-sparse factor case.

Intuitively, reducing modulo a 2-sparse factor $g(\mathtt{X})$ of degree n with $n = N/m$, the dimension is reduced from N to $n = N/m$ and the noise weight is reduced from $2t$ to a value between $2t/m$ and $2t$ because some error positions are overlapping and vanishing. Now we consider the reduced instance and then apply existing attacks.

Assume that each of the t errors in e is independently and uniformly sampled. Then each error of e lands on a random position in $\bar{e} := e \mod g(\mathtt{X})$ of length n. For $j \in [n]$, define the E_j the event that j-th position in the reduced polynomial $\bar{e}$ equals to 0. Then $\Pr[E_j] = (1 - 1/n)^t$. Thus, the expected Hamming weight of errors in the reduced Ring-LPN instance is $2n(1 - (1 - 1/n)^t)$. In summary the new reduced LPN instance has dimension n, sample number $2n$, and the expected number of errors $w_n = 2n(1 - (1 - 1/n)^t)$. Then, we have the following attack costs.

Gaussian Elimination. Via iterating over all nontrivial factors of N, the cost of the Gaussian elimination attack is

$$O\left(\min_{n|N}\left(\frac{1}{1 - w_n/(2n)}\right)^n \cdot n^2 \log n\right).$$

Information Set Decoding. When reducing a factor $g(\mathtt{X})$ of $f(\mathtt{X})$, the assumption that the error weight is much smaller than the dimension of the code does not hold any more. Via choosing appropriate factor n of N, it is possible that there exists a reduced instance, such that some ISD variants outperform Prange's ISD decoding algorithm. To this end, we apply the lower bound results of [HOSS18].

Roughly speaking, the complexity of ISD algorithms for a parity check matrix H with dimension n, sample number q, and error weight w_n is lower bounded by

$$\min_{p_0,p_1}\left\{\frac{\min\left\{2^n, \binom{q}{w_n}\right\}}{\binom{n-p_1}{w_n-p_0}} \cdot \left(\frac{K_1 + K_2}{\binom{n+p_1}{p_0}} + \frac{w \cdot (n - p_1)}{2^{p_1}}\right)\right\},$$

where (p_0, p_1) satisfy $0 \le p_1 \le q - n$ and $0 \le p_0 \le n + p_1$, K_1 is the cost of performing a Gaussian elimination on a submatrix of H with $n - p_1$ columns, K_2 is the complexity of a special sub-algorithm. From [HOSS18], K_2 is lower

bounded by

$$\binom{(n+p_1)/2}{p/8}$$

if the algorithm of [BJMM12] is used. As the parity check matrix H has the quasi-cyclic structure, we assume that the Gaussian elimination can be done in time $(n-p_1)^2 \log(n-p_1)$. Combining everything, a lower bound of cost of the the ISD algorithms is

$$\min_{n|N} \left\{ \frac{\min\left\{2^n, \binom{q}{w_n}\right\}}{\binom{n-p_1}{w_n-p_0}} \cdot \left(\frac{(n-q)^2 \log(n-q) + \binom{(n+p_1)/2}{p/8}}{\binom{n+p_1}{p_0}} + \frac{w \cdot (n-p_1)}{2^{p_1}} \right) \right\}.$$

Statistical Decoding. Similarly, the cost of the SD attack is

$$O\left(\min_{n|N} \left(\frac{q}{n-1} \right)^{w_n} \cdot (q-n) \right).$$

To withstand the above attacks for Ring-LPN, one method is to let N be a prime so that $\mathbb{G} = \mathbb{Z}_N$ has no nontrivial subgroups. In this case, the regular noise structure can not be used since the noise weight w never divides a prime N. Hence, the optimized PCG seed size for regular distributions does not apply to prime N. Another option is to choose N being the largest prime factor of $p^k - 1$. As for binary correlations, we suggest k of $\mathbb{F}_{2^k}$ s.t. $2^k - 1$ is a Mersenne prime.

Folding Attacks for QA-SD. Assume $\mathcal{R} = \mathbb{F}_{p^k}[\mathtt{X}_1, \ldots, \mathtt{X}_n]/(\mathtt{X}_1^d - 1, \ldots, \mathtt{X}_n^d - 1)$, where $d \mid (p^k - 1)$ and $G = \mathbb{Z}_d^n$. Given a QA-SD instance (a, b) over $\mathcal{R}$, an adversary can exploit the quasi-abelian structure to construct a reduced QA-SD instance with smaller code length and dimension via modulo a subgroup $H \leq G$. As noted in [BCCD23, BBC+24], the most effective attack, termed *folding attack*, leverages such subgroups. The folding attack maps a linear code of length $2|G|$ to one of length $2|G/H|$, preserving the code rate. This operation is applied independently to each block, ensuring the number of errors per block remains bounded by t. For $p = 2, k = 2, d = 3$, and $G = \mathbb{Z}_3^n$, there are $\begin{bmatrix} n \\ \eta \end{bmatrix}_3$ subgroups of cardinality 3^η, where $\begin{bmatrix} n \\ \eta \end{bmatrix}_3$ is the Gaussian binomial coefficient representing the number of η-dimensional subspaces of $\mathbb{Z}_3^n$ [And74].

Parameters are chosen such that ISD variants against all folded instances meet the desired security level. Folding attacks are effective only for large subgroups H, as these produce smaller folded instances of the decoding problem. For a folding attack to succeed, the folded instance must have a unique solution, requiring the number of errors in the folded instance to be below its Gilbert-Varshamov (GV) bound. Finding a solution for any folded instance below the GV bound suffices to compromise the QA-SD instance.

The folded error vector's weight closely approximates that of the initial vector with overwhelming probability due to the sparsity of the error distribution. A formal analysis will show that the probability of the folded error having a much smaller weight $\widetilde{w}$ is exponentially small ($\Pr_{\widetilde{w}}$). Exploiting the large number of subgroups H in G, the adversary performs folding attacks to decode errors of

weight $\widetilde{w}$. This approach, as noted in [BBC+24], was used to argue that the security parameters in [BCCD23] are overestimated.

Let $H \preceq G$. The group homomorphism $\pi_H : G \to G/H$ extends to an morphism of algebra:

$$\begin{aligned} \pi_H : \quad \mathbb{F}_{p^k}[G] \quad &\longrightarrow \quad \mathbb{F}_{p^k}[G/H] \\ \sum_{g\in G} a_g \cdot g \quad &\longmapsto \sum_{\overline{g}\in G/H} \left(\sum_{h\in H} a_{h+\overline{g}}\right) \overline{g} \end{aligned}$$

This maps a vector of length $|G|$ to one of length $|G/H|$, summing the $|H|$ entries of each coset of G/H in G. For $(a, b) \in \mathbb{F}_{p^k}[G]^2$, the folding operation π_H is extended as

$$\pi_H(a, b) := (\pi_H(a), \pi_H(b)) \in \mathbb{F}_{p^k}[G/H]^2.$$

For a QA-SD instance (a, b), the folding operation π_H maps

$$\pi_H\left(a \cdot s + e\right) = \pi_H(a) \cdot \pi_H(s) + \pi_H(e) \in \mathbb{F}_{p^k}[G/H],$$

producing the syndrome of $\pi_H(e, s)$ with respect to the matrix induced by $\pi_H(1, a)$. The goal is to estimate the weight of $\pi_H(e, s)$, where e and s are independent and uniformly distributed with Hamming weight t.

Let $\ell := |H|$ and $\mathcal{C}_\ell := [\ell, \ell - 1]_q$ be the parity code. Denote $P_{\nu,\ell}(\mathtt{X})$ as the weight enumerator of $\mathcal{C}_\ell$ and $P_{\theta,\ell}(\mathtt{X})$ as the weight enumerator of $\mathbb{F}_q^\ell \backslash \mathcal{C}_\ell$ [MS86, Chapter 5.2].

Lemma 9.1. *[BBC+24, Proposition 18] Given $t \in [0, |G|]$ and $u \in [0, \min(t, |G/H|)]$, for e uniformly sampled over the Hamming weight t elements of $\mathbb{F}_{p^k}[G]$, it holds that*

$$\Pr_e[\mathsf{wt}(\pi_H(e)) = u] = \frac{\binom{|G/H|}{u} \cdot [\mathtt{X}^t]\left(P^u_{\theta,\ell}(\mathtt{X}) P^{|G/H|-u}_{\nu,\ell}(\mathtt{X})\right)}{\binom{|G|}{t}(q-1)^t},$$

where $\binom{|G/H|}{u} \cdot [\mathtt{X}^t]\left(P^u_{\theta,\ell}(\mathtt{X}) P^{|G/H|-u}_{\nu,\ell}(\mathtt{X})\right)$ is defined as the coefficient of $\mathtt{X}^t$ in the polynomial $\left(\binom{|G/H|}{u} P^u_{\theta,\ell}(\mathtt{X}) P^{|G/H|-u}_{\nu,\ell}(\mathtt{X})\right)$.

MacWilliam's identity gives the formulae for $P_{\nu,\ell}$ and $P_{\theta,\ell}$.

Lemma 9.2. *[CT19, Lemma 1] It holds that*

$$P_{\nu,\ell}(\mathtt{X}) := \frac{1}{q}\left((1 + (q-1)\mathtt{X})^\ell + (q-1)(1-\mathtt{X})^\ell\right)$$

and

$$P_{\theta,\ell}(\mathtt{X}) := \frac{q-1}{q}\left((1 + (q-1)\mathtt{X})^\ell - (1-\mathtt{X})^\ell\right).$$

Lemma 9.1 shows that the probability depends only on the cardinality of H.

The subgroup H is selected such that $t/|G/H|$ is below the GV bound of the folded code of rate 1/2. For the QA-SD assumption with $p = 2, k = 2, d = 3$, $G = \mathbb{Z}_3^n$ and $H = \mathbb{Z}_3^\eta$, it holds that $|G/H| = 3^{n-\eta}$. Specifically, $t \leq \delta_{GV} 3^{n-\eta}$. Choosing $\eta = n - \lceil \log_3(t/\delta_{GV}) \rceil$ or $\eta = n - \lceil \log_3(t/\delta_{GV}) \rceil - 1$ minimizes the complexity.

An adversary may assume a smaller error weight τ for the folded instance, run the best decoding algorithm, and abort if the operation count exceeds the threshold for decoding τ errors. The adversary then asserts that the folded instance has no error vector below τ and repeats the process with another random subgroup H. The average runtime is

$$\frac{C_{\mathsf{Dec}}(\tau) + C_{\mathsf{fold}}}{\Pr[\mathsf{wt}(\pi_H(\mathbf{e})) = \tau]},$$

where $C_{\mathsf{Dec}}(\tau)$ is the complexity of the best decoding algorithm for weight $\leq \tau$ and C_{fold} is the complexity of the folding operation, which scales with the original code length. We choose τ to minimize the total complexity. Security parameters are selected using a SageMath script [Bom24].

Algebraic Decoding Attacks. To utilize the structure of the linear codes, the algebraic decoding attacks [Pel92, Kot92] were proposed to attack the McEliece cryptosystem employing various hiding structures. The rough idea is to distinguish the component-wise products of the target code from the component-wise products of random linear codes. There is a line of works of algebraic decoding attacks breaking the McEliece cryptosystem in the literature [MCMMP11, FGO+13, CMP14]. Algebraic decoding attacks could distinguish the Ring-LPN or QA-SD instances from random instances only if the given generator matrix $G \in \mathbb{F}_q^{(n-k)\times n}$ of the linear code is strongly multiplicative, i.e., the pairwise tensor products of the columns of G span a new linear code with dimension strictly less than $(n-k)^2$. In our scenario, given a random linear matrix G derived from H, G being strongly multiplicative holds with negligible probability because the tensor products of arbitrary two columns are linearly independent with overwhelming probability. Hence, the algebraic decoding attacks do not work in our setting.

Decoding One Out of Many Attacks. For a given syndrome decoding problem with M distinct syndromes, the decoding one out of many (DOOM) attack [Sen11] provides a $\sqrt{M}$ factor acceleration for the ISD algorithms. For Ring-LPN/QA-SD assumptions, one can derive N new syndrome decoding instances due to the quasi-cyclic or quasi-abelian structure. Specifically, given an instance $(a, b = a \cdot e_0 + e_1)$, the N instances associated to the original instance (a, b) are $(a, b \cdot \vec{\mathtt{X}}^i)$ for $i \in [0, N-1]$ where $\{\vec{\mathtt{X}}^i\}_{i\in[0,N-1]}$ is a basis of the ring $\mathcal{R}$. If the polynomial $f(\mathtt{X}) := \mathtt{X}^N - 1$ in Ring-LPN has a non-trivial 2-sparse factor $g(\mathtt{X})$ of degree d, the DOOM attack provides a factor $\sqrt{d}$ speedup to the ISD algorithms on the reduced Ring-LPN instance with dimension d.

We observe that in our setting for $k > 1$, given an instance $(a, b = a \cdot e_0 + e_1)$, there is a different method to exploit the structure of $\mathbb{F}_{p^k}$ to derive new instances. In particular, for $j \in [0, k-1]$, each $(a^{p^j}, b^{p^j} = a^{p^j} \cdot e_0^{p^j} + e_1^{p^j})$ is a valid instance as

each of $(e_0^{p^j}, e_1^{p^j})$ has weight t as well. Combining the quasi-abelian or the quasi-cyclic structure with the above observation, the DOOM provides an acceleration factor $\sqrt{N \cdot k}$ to the ISD attacks.

Attack on the Assumptions with Static Leakage. The malicious PCG setup protocols rely on the Ring-LPN or QA-SD with static leakage. Recall that in the static leakage security game, the adversary is able to query $2t$ predicates $P_j^i : [0, N-1] \to \{0,1\}$ once the error vector (e_0, e_1) is sampled, where P_j^i takes the position of j-th non-zero value of e_i. If one of $\{P_j^i\}_{i,j}$ evaluates to 0, the game aborts and the adversary loses the security game. While if all of $\{P_j^i\}_{i,j}$ evaluate to 1, the game sends $\mathsf{success}$ to the adversary. Based on the query results, the adversary tries to distinguish between a valid Ring-LPN or QA-SD instance and a random instance. We observe that for all of the attacks described in this section, the decisions made by the adversary to distinguish $(a, b = a \cdot e_0 + e_1)$ from a uniform instance are independent from b. For instance, the Gaussian elimination attack only guesses the error free positions and the statistical decoding attack finds the small weight codewords. Hence, the distinguishing probability of the adversary could be estimated by hybrid arguments, where the error vector is sampled after the adversary makes the decisions. Therefore, for the same parameters, Ring-LPN or QA-SD with static leakage provides the same security level as Ring-LPN or QA-SD without leakage.

9.2 Parameters

According to the above mentioned attacks in Section 9.1, we show various parameters for Ring-LPN assumptions and the QA-SD assumptions in our PCG constructions. In particular, we consider the most interesting case of OLEs over $\mathbb{F}_2$ and authenticated Boolean triples. Recall that in the Ring-LPN based PCG constructions, N is the vector length satisfying $N \mid 2^k - 1$ and t denotes the Hamming weight of each length N error vector. In the QA-SD based PCG constructions, $d \mid 2^k - 1$ and $N = d^n$, where n denotes the number of variables.

To enable faster SPFSS evaluations (approximately $t\times$ savings in computation), one can assume the error vector has additionally a regular noise. This additionally requires $t \mid N$, hence limits the parameter choices. To generate appropriate parameters, we use the assumption of the form

$$(a_0, a_1 \dots a_c, b = \sum_{i \in [0, c-1]} a_i \cdot e_i),$$

as well as in [BCG+20, BCCD23, BBC+24], where $a_0 = 1$, $a_i \xleftarrow{\$} \mathcal{R}$ and $e_i \xleftarrow{\$} \mathcal{R}$ of t-sparsity. Here c is denote as a *syndrome compression* parameter. Thus, ct is the actual error weight of the syndrome decoding problem. Recall that for simplicity, in our constructions, the parameter c is set as 2, i.e., $(1, a_1, b = e_0 + a_1 \cdot e_1)$, and it is straightforward to extend our constructions to $c \geq 2$. In the following, we summarize key sizes of our PCG constructions, taking $c \geq 2$ into consideration (Basically, the formulas are obtained by just replacing t with $ct/2$ from those in the corresponding theorems, e.g. Theorem 5.1).

PCG seed size from Ring-LPN/QA-SD:

1. Regular noise:
   - OLEs over $\mathbb{F}_2$: $k(ct)^2\big((\log N - \log t + 1)(\lambda+2) + \lambda + k\big) + ct(\log N + k)$.
   - Authenticated Boolean triples: $(k^2(ct)^2 + k(ct)^2 + 2kct + ct)\big((\log N - \log t + 1)(\lambda+2) + \lambda\big) + \eta(k^2(ct)^2 + 2kct) + k(k(ct)^2 + 2ct) + \eta$.
2. Non-Regular noise:
   - OLEs over $\mathbb{F}_2$: $k(ct)^2\big(\log N(\lambda+2) + \lambda + k\big) + ct(\log N + k)$.
   - Authenticated Boolean triples: $(k^2(ct)^2 + k(ct)^2 + 2kct + 2ct)\big(\log N(\lambda+2) + \lambda\big) + \eta(k^2(ct)^2 + 2kct) + k(k(ct)^2 + 2ct) + \eta$.

Remark 9.1. The concrete seed sizes of constructions from Ring-LPN and QA-SD vary only due to the different restrictions of k.

Error type	λ	c	t	$\mathbb{F}_{2^k}$	N	Seed size	kN OLEs	Stretch
Regular	80	3	29	$\mathbb{F}_{2^{28}}$	$2^{28}-1$	$2^{28.72}$	$2^{32.80}$	16.98
Non-regular	80	2	43	$\mathbb{F}_{2^{31}}$	$2^{31}-1$	$2^{29.18}$	$2^{35.95}$	109.44
Non-regular	80	3	29	$\mathbb{F}_{2^{31}}$	$2^{31}-1$	$2^{29.21}$	$2^{35.95}$	106.94
Regular	128	5	29	$\mathbb{F}_{2^{28}}$	$2^{28}-1$	$2^{30.85}$	$2^{32.80}$	3.87
Regular	128	4	33	$\mathbb{F}_{2^{32}}$	$2^{32}-1$	$2^{30.97}$	$2^{36.99}$	64.96
Regular	128	2	71	$\mathbb{F}_{2^{35}}$	$2^{35}-1$	$2^{31.41}$	$2^{40.12}$	421.41
Non-regular	128	2	65	$\mathbb{F}_{2^{31}}$	$2^{31}-1$	$2^{31.03}$	$2^{35.95}$	30.33
Non-regular	128	4	33	$\mathbb{F}_{2^{31}}$	$2^{31}-1$	$2^{31.07}$	$2^{35.95}$	29.42

Table 4. Seed size for OLEs over $\mathbb{F}_2$ from Ring-LPN.

Error Type	λ	c	t	$\mathbb{F}_{2^k}$	N	Seed Size	kN Triples	Stretch
Regular	80	3	29	$\mathbb{F}_{2^{28}}$	$2^{28}-1$	$2^{33.61}$	$2^{32.80}$	0.57
Regular	128	4	33	$\mathbb{F}_{2^{32}}$	$2^{32}-1$	$2^{36.05}$	$2^{36.99}$	1.92
Regular	128	2	71	$\mathbb{F}_{2^{35}}$	$2^{35}-1$	$2^{36.61}$	$2^{40.12}$	11.41
Non-regular	80	2	43	$\mathbb{F}_{2^{31}}$	$2^{31}-1$	$2^{34.21}$	$2^{35.95}$	3.34
Non-regular	80	3	29	$\mathbb{F}_{2^{31}}$	$2^{31}-1$	$2^{34.24}$	$2^{35.95}$	3.26
Non-regular	128	2	66	$\mathbb{F}_{2^{31}}$	$2^{31}-1$	$2^{36.10}$	$2^{35.95}$	0.89

Table 5. Seed size for authenticated Boolean triples from Ring-LPN.

We remark that as shown in the tables, the efficiency of our constructions is comparable to the constructions of [BCG+20, BCCD23, BBC+24]. The QA-SD assumption provides more freedom to choose parameters than the Ring-LPN assumption while the Ring-LPN assumption employs well-studied FFT techniques. In particular, for Ring-LPN assumption, the parameter k of the field $\mathbb{F}_{2^k}$ determines the upper bound of the number of PCGs. Whereas QA-SD has the additional parameter n of the number of variables to adjust the number of correlations, allowing arbitrary stretch.

Error Type	λ	c	t	$\mathbb{F}_{2^k}$	n	N	Seed Size	kN OLEs	Stretch
Regular	80	3	27	$\mathbb{F}_{2^2}$	16	3^{16}	$2^{24.53}$	$2^{26.35}$	3.53
Regular	80	9	9	$\mathbb{F}_{2^2}$	19	3^{19}	$2^{24.89}$	$2^{31.11}$	74.63
Regular	80	2	49	$\mathbb{F}_{2^3}$	10	7^{10}	$2^{25.78}$	$2^{29.65}$	14.65
Regular	80	2	49	$\mathbb{F}_{2^3}$	12	7^{12}	$2^{26.08}$	$2^{35.27}$	584.15
Regular	80	6	15	$\mathbb{F}_{2^4}$	7	15^{7}	$2^{26.01}$	$2^{29.34}$	10.10
Regular	80	6	15	$\mathbb{F}_{2^4}$	8	15^{8}	$2^{26.21}$	$2^{33.25}$	131.36
Non-regular	80	2	42	$\mathbb{F}_{2^2}$	16	3^{16}	$2^{24.86}$	$2^{26.35}$	2.82
Non-regular	80	2	42	$\mathbb{F}_{2^2}$	18	3^{18}	$2^{25.02}$	$2^{29.52}$	22.67
Non-regular	80	5	17	$\mathbb{F}_{2^2}$	19	3^{19}	$2^{25.13}$	$2^{31.11}$	63.04
Non-regular	80	5	17	$\mathbb{F}_{2^2}$	20	3^{20}	$2^{25.20}$	$2^{32.69}$	179.97
Regular	128	2	81	$\mathbb{F}_{2^2}$	19	3^{19}	$2^{27.38}$	$2^{31.11}$	13.21
Regular	128	5	27	$\mathbb{F}_{2^2}$	19	3^{19}	$2^{26.95}$	$2^{31.11}$	17.92
Regular	128	5	27	$\mathbb{F}_{2^2}$	21	3^{21}	$2^{27.10}$	$2^{34.28}$	144.61
Regular	128	3	49	$\mathbb{F}_{2^3}$	10	7^{10}	$2^{27.61}$	$2^{29.65}$	4.10
Regular	128	3	49	$\mathbb{F}_{2^3}$	12	7^{12}	$2^{27.91}$	$2^{35.27}$	163.79
Regular	128	9	15	$\mathbb{F}_{2^4}$	7	15^{7}	$2^{27.84}$	$2^{29.34}$	2.83
Regular	128	9	15	$\mathbb{F}_{2^4}$	8	15^{8}	$2^{28.05}$	$2^{33.25}$	36.83
Non-regular	128	2	66	$\mathbb{F}_{2^2}$	17	3^{17}	$2^{26.91}$	$2^{27.94}$	2.04
Non-regular	128	4	33	$\mathbb{F}_{2^2}$	18	3^{18}	$2^{26.99}$	$2^{29.52}$	5.79
Non-regular	128	4	33	$\mathbb{F}_{2^2}$	19	3^{19}	$2^{27.07}$	$2^{31.11}$	16.49
Non-regular	128	4	33	$\mathbb{F}_{2^2}$	20	3^{20}	$2^{27.14}$	$2^{32.69}$	47.07

Table 6. Seed size for OLEs over $\mathbb{F}_2$ from QA-SD.

Error Type	λ	c	t	$\mathbb{F}_{2^k}$	n	N	Seed Size	kN Triples	Stretch
Regular	80	3	27	$\mathbb{F}_{2^2}$	16	3^{16}	$2^{26.17}$	$2^{26.35}$	1.13
Regular	80	3	27	$\mathbb{F}_{2^2}$	19	3^{19}	$2^{26.44}$	$2^{31.11}$	25.41
Regular	80	9	9	$\mathbb{F}_{2^2}$	20	3^{20}	$2^{26.60}$	$2^{32.69}$	68.47
Non-regular	80	2	43	$\mathbb{F}_{2^2}$	18	3^{18}	$2^{26.72}$	$2^{29.52}$	6.97
Non-regular	80	2	43	$\mathbb{F}_{2^2}$	19	3^{19}	$2^{26.80}$	$2^{31.11}$	19.88
Non-regular	80	2	43	$\mathbb{F}_{2^2}$	20	3^{20}	$2^{26.87}$	$2^{32.69}$	56.83
Regular	128	2	81	$\mathbb{F}_{2^2}$	19	3^{19}	$2^{29.01}$	$2^{31.11}$	4.27
Regular	128	5	27	$\mathbb{F}_{2^2}$	20	3^{20}	$2^{28.65}$	$2^{32.69}$	16.45
Regular	128	5	27	$\mathbb{F}_{2^2}$	21	3^{21}	$2^{28.73}$	$2^{34.28}$	46.85
Non-regular	128	2	66	$\mathbb{F}_{2^2}$	18	3^{18}	$2^{28.62}$	$2^{29.52}$	1.87
Non-regular	128	2	66	$\mathbb{F}_{2^2}$	19	3^{19}	$2^{28.69}$	$2^{31.11}$	5.34
Non-regular	128	2	66	$\mathbb{F}_{2^2}$	20	3^{20}	$2^{28.76}$	$2^{32.69}$	15.27

Table 7. Seed size for authenticated Boolean triples from QA-SD.

9.3 Performance Evaluation

We implement our PCGs [7] based on the source code of $\mathbb{F}_4$OLEAGE [BBC+24, Bom24]. We then run the programs on a PC with Intel(R) Xeon(R) Gold 5220R 2.20GHz CPU and 128GB of RAM.

For fairness, we select the same concrete parameters and evaluate the efficiency of generating binary triples in the same way as in $\mathbb{F}_4$OLEAGE. The results are reported in Table 3. Specifically, we take $c = 3$, $t = 27$, $n = 16$ and $\mathbb{F}_4$, which allows for generating $2 \cdot 3^{16}$ OLEs over $\mathbb{F}_2$ by our approach. Suppose the time is T. We then estimate the per-party cost to generate 10^9 Beaver triples as $T \cdot \frac{10^9}{2 \cdot 3^{16}}$ for the two-party case, and $2 \cdot (10-1) \cdot T \cdot \frac{10^9}{2 \cdot 3^{16}}$ for the 10-party case. As for communication, we compute an estimate of $C \approx 26$MB of communication for generating a seed for $2 \cdot 3^{18}$ OLEs from using the distributed protocol of $\mathbb{F}_4$OLEAGE. For 10^9 two-party Beaver triples, the per-party communication is estimated as $C \cdot \frac{10^9}{2 \cdot 3^{18}}$, while for the 10-party case, the communication would be $2 \cdot (10-1) \cdot C \cdot \frac{10^9}{2 \cdot 3^{18}}$. In general, the speed of our triple generation is roughly $2\times$ slower than $\mathbb{F}_4$OLEAGE, while our communication is almost the same as $\mathbb{F}_4$OLEAGE. Note that $\mathbb{F}_4$OLEAGE requires additional linear communication for seed expansion in the multi-party case.

To evaluate the seed expansion time for generating authenticated Boolean multiplication triples, we additionally implement the FFT over $\mathbb{F}_{2^{128}}, \mathbb{F}_{2^{64}}$. The results are reported in Table 2. Concretely, the speed of our triple generation is derived from running our program on our machine with $c = 3$, $t = 27$, $n = 16$, $\mathbb{F}_4$ and 128-bit MAC key. While the communication cost is estimated for generating 3^{18} authenticated triples. In summary, our approach for authenticated Boolean triples with 128-bit MAC key has similar efficiency as [BCG+20] for authenticated triples over a 128-bit prime field.

Acknowledgments. The authors would like to thank Yuval Ishai for his many helpful discussions and valuable suggestions on this work. They also thank Yi Kuang and Wenhao Zhang for their assistance in running the programs. Additionally, the authors are grateful for the insightful comments from the anonymous reviewers. This work was supported in part by the National Key Research and Development (R&D) Program of China under Grants 2022YFA1004900, in part by the National Natural Science Foundation of China under Grants 12361141818, 12426302, 12031011, and 12271084, and in part by the Natural Science Foundation of Shanghai under the 2024 Shanghai Action Plan for Science, Technology, and Innovation Grant 24BC3200700.

References

ABG+24. Amit Agarwal, Elette Boyle, Niv Gilboa, Yuval Ishai, Mahimna Kelkar, and Yiping Ma. Compressing unit-vector correlations via sparse pseudo-

[7] The code is available at https://github.com/zhli271828/Trace-F2-OLE-PCG.

random generators. In *CRYPTO 2024*, volume 14927 of *LNCS*, pages 346–383. Springer, 2024.

And74. George E. Andrews. Applications of basic hypergeometric functions. *SIAM Review*, 16(4):441–484, 1974.

BBC+24. 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 *ASIACRYPT 2024*, volume 15489 of *LNCS*, pages 69–101. Springer, 2024.

BCCD23. Maxime Bombar, Geoffroy Couteau, Alain Couvreur, and Clément Ducros. Correlated pseudorandomness from the hardness of quasi-abelian decoding. In *CRYPTO 2023*, volume 14084 of *LNCS*, pages 567–601. Springer, 2023.

BCG+19a. 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 *CCS 2019*, pages 291–308. ACM, 2019.

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

BCG+20. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators from ring-lpn. In *CRYPTO 2020*, volume 12171 of *LNCS*, pages 387–416. Springer, 2020.

BCG+22. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Nicolas Resch, and Peter Scholl. Correlated pseudorandomness from expand-accumulate codes. In *CRYPTO 2022*, volume 13508 of *LNCS*, pages 603–633. Springer, 2022.

BCG+23. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Nicolas Resch, and Peter Scholl. Oblivious transfer with constant computational overhead. In *EUROCRYPT 2023*, volume 14004 of *LNCS*, pages 271–302. Springer, 2023.

BCGI18. Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In *CCS 2018*, pages 896–912. ACM, 2018.

Bea91. Donald Beaver. Efficient multiparty protocols using circuit randomization. In *CRYPTO '91*, volume 576 of *LNCS*, pages 420–432. Springer, 1991.

BFKL93. Avrim Blum, Merrick L. Furst, Michael J. Kearns, and Richard J. Lipton. Cryptographic primitives based on hard learning problems. In *CRYPTO 1993*, volume 773 of *LNCS*, pages 278–291. Springer, 1993.

BGI15. Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In *EUROCRYPT 2015*, volume 9057 of *LNCS*, pages 337–367. Springer, 2015.

BGI16. Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, 2016*, pages 1292–1303. ACM, 2016.

BJMM12. Anja Becker, Antoine Joux, Alexander May, and Alexander Meurer. Decoding random binary linear codes in $2^{n/20}$: How $1 + 1 = 0$ improves information set decoding. In *EUROCRYPT*, volume 7237 of *LNCS*, pages 520–536. Springer, 2012.

BKW00. Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. In *STOC*, pages 435–440. ACM, 2000.

BLP11. Daniel J. Bernstein, Tanja Lange, and Christiane Peters. Smaller decoding exponents: Ball-collision decoding. In *CRYPTO*, volume 6841 of *LNCS*, pages 743–760. Springer, 2011.

BM18. Leif Both and Alexander May. Decoding linear codes with high error rate and its impact for LPN security. In *PQCrypto*, volume 10786 of *LNCS*, pages 25–46. Springer, 2018.

BNO19. Aner Ben-Efraim, Michael Nielsen, and Eran Omri. Turbospeedz: Double your online spdz! improving SPDZ using function dependent preprocessing. In *Applied Cryptography and Network Security, ACNS 2019*, volume 11464 of *LNCS*, pages 530–549. Springer, 2019.

Bom24. Maxime Bombar. Estimator complexity for the $\mathbb{F}_4$OLEAGE PCG. https://github.com/mbombar/estimator_folding, 2024.

BP06. Joan Boyar and René Peralta. Concrete multiplicative complexity of symmetric functions. In *MFCS*, volume 4162 of *LNCS*, pages 179–189. Springer, 2006.

CMP14. Irene Marquez Corbella, Edgar Martínez-Moro, and Ruud Pellikaan. On the unique representation of very strong algebraic geometry codes. *Des. Codes Cryptogr.*, 70(1-2):215–230, 2014.

Cou19. Geoffroy Couteau. A note on the communication complexity of multiparty computation in the correlated randomness model. In *EUROCRYPT 2019*, volume 11477 of *LNCS*, pages 473–503. Springer, 2019.

CT19. Rodolfo Canto-Torres and Jean-Pierre Tillich. Speeding up decoding a code with a non-trivial automorphism group up to an exponential factor. In *ISIT*, pages 1927–1931. IEEE, 2019.

DEEK24. Léo Ducas, Andre Esser, Simona Etinski, and Elena Kirshanova. Asymptotics and improvements of sieving for codes. In *EUROCRYPT (6)*, volume 14656 of *LNCS*, pages 151–180. Springer, 2024.

DKL$^+$13. Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In *Computer Security - ESORICS 2013 - 18th European Symposium on Research in Computer Security, 2013*, volume 8134 of *LNCS*, pages 1–18. Springer, 2013.

DNNR17. Ivan Damgård, Jesper Buus Nielsen, Michael Nielsen, and Samuel Ranellucci. The tinytable protocol for 2-party secure computation, or: Gate-scrambling revisited. In *CRYPTO 2017*, volume 10401 of *Lecture Notes in Computer Science*, pages 167–187. Springer, 2017.

DPSZ12. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, 2012.

EGP$^+$23. Daniel Escudero, Vipul Goyal, Antigoni Polychroniadou, Yifan Song, and Chenkai Weng. Superpack: Dishonest majority MPC with constant online communication. In *EUROCRYPT 2023*, volume 14005 of *LNCS*, pages 220–250. Springer, 2023.

EGPS22. Daniel Escudero, Vipul Goyal, Antigoni Polychroniadou, and Yifan Song. Turbopack: Honest majority MPC with constant online communication. In *CCS 2022*, pages 951–964. ACM, 2022.

EKM17. Andre Esser, Robert Kübler, and Alexander May. LPN decoded. In *CRYPTO (2)*, volume 10402 of *LNCS*, pages 486–514. Springer, 2017.

EXY22. Daniel Escudero, Chaoping Xing, and Chen Yuan. More efficient dishonest majority secure computation over $\mathbb{Z}_{2^k}$ via galois rings. In *CRYPTO (1)*, volume 13507 of *LNCS*, pages 383–412. Springer, 2022.

FGO+13. Jean-Charles Faugère, Valérie Gauthier-Umaña, Ayoub Otmani, Ludovic Perret, and Jean-Pierre Tillich. A distinguisher for high-rate mceliece cryptosystems. *IEEE Trans. Inf. Theory*, 59(10):6830–6844, 2013.

FS09. Matthieu Finiasz and Nicolas Sendrier. Security bounds for the design of code-based cryptosystems. In *ASIACRYPT*, volume 5912 of *LNCS*, pages 88–105. Springer, 2009.

GI14. Niv Gilboa and Yuval Ishai. Distributed point functions and their applications. In *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 640–658. Springer, 2014.

GYW+23. Xiaojie Guo, Kang Yang, Xiao Wang, Wenhao Zhang, Xiang Xie, Jiang Zhang, and Zheli Liu. Half-tree: Halving the cost of tree expansion in COT and DPF. In *EUROCRYPT (1)*, volume 14004 of *LNCS*, pages 330–362. Springer, 2023.

HOSS18. Carmit Hazay, Emmanuela Orsini, Peter Scholl, and Eduardo Soria-Vazquez. Tinykeys: A new approach to efficient multi-party computation. In *CRYPTO (3)*, volume 10993 of *LNCS*, pages 3–33. Springer, 2018.

KOS16. Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: faster malicious arithmetic secure computation with oblivious transfer. In *CCS 2016*, pages 830–842. ACM, 2016.

Kot92. Ralf Kotter. An unified description of an error locating procedure for linear codes. *Proc. IAACCT, Voneshta Voda, Bulgaria*, 1992.

KPR18. Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: Making SPDZ great again. In *EUROCRYPT (3)*, volume 10822 of *Lecture Notes in Computer Science*, pages 158–189. Springer, 2018.

LAH16. Sian-Jheng Lin, Tareq Y. Al-Naffouri, and Yunghsiang S. Han. FFT algorithm for binary extension finite fields and its application to reed-solomon codes. *IEEE Trans. Inf. Theory*, 62(10):5343–5358, 2016.

LN97. Rudolf Lidl and Harald Niederreiter. *Finite Fields*. Cambridge University Press, 1997.

LWYY24. Hanlin Liu, Xiao Wang, Kang Yang, and Yu Yu. The hardness of LPN over any integer ring and field for PCG applications. In *EUROCRYPT 2024*, volume 14656 of *LNCS*, pages 149–179. Springer, 2024.

LXYZ24. Hongqing Liu, Chaoping Xing, Chen Yuan, and Taoxu Zou. Dishonest majority multiparty computation over matrix rings. In *ASIACRYPT 2024*, volume 15489 of *LNCS*, pages 299–327. Springer, 2024.

Lyu05. Vadim Lyubashevsky. The parity problem in the presence of noise, decoding random linear codes, and the subset sum problem. In *APPROX-RANDOM*, volume 3624 of *LNCS*, pages 378–389. Springer, 2005.

MCMMP11. Irene Márquez-Corbella, Edgar Martınez-Moro, and Ruud Pellikaan. Evaluation of public-key cryptosystems based on algebraic geometry codes. In *Proceedings of the Third International Castle Meeting on Coding Theory and Applications*, pages 199–204, 2011.

MMT11. Alexander May, Alexander Meurer, and Enrico Thomae. Decoding random linear codes in $\tilde{O}(2^{0.054n})$. In *ASIACRYPT*, volume 7073 of *LNCS*, pages 107–124. Springer, 2011.

MO15. Alexander May and Ilya Ozerov. On computing nearest neighbors with applications to decoding of binary linear codes. In *EUROCRYPT (1)*, volume 9056 of *LNCS*, pages 203–228. Springer, 2015.

MS86. F.J. MacWilliams and N.J.A. Sloane. *The Theory of Error-Correcting Codes*. North-holland Publishing Company, 2nd edition, 1986.

Obe07. Ulrich Oberst. The fast fourier transform. *SIAM J. Control. Optim.*, 46(2):496–540, 2007.

Pel92. Ruud Pellikaan. On decoding by error location and dependent sets of error positions. *Discret. Math.*, 106-107:369–381, 1992.

Pra62. Eugene Prange. The use of information sets in decoding cyclic codes. *IRE Trans. Inf. Theory*, 8(5):5–9, 1962.

PSSY21. Arpita Patra, Thomas Schneider, Ajith Suresh, and Hossein Yalame. ABY2.0: improved mixed-protocol secure two-party computation. In *USENIX Security Symposium*, pages 2165–2182. USENIX Association, 2021.

Roy22. Lawrence Roy. Softspokenot: Quieter OT extension from small-field silent VOLE in the minicrypt model. In *CRYPTO (1)*, volume 13507 of *LNCS*, pages 657–687. Springer, 2022.

RRT23. Srinivasan Raghuraman, Peter Rindal, and Titouan Tanguy. Expand-convolute codes for pseudorandom correlation generators from LPN. In *CRYPTO 2023*, volume 14084 of *LNCS*, pages 602–632. Springer, 2023.

RS22. Rahul Rachuri and Peter Scholl. Le mans: Dynamic and fluid MPC for dishonest majority. In *CRYPTO 2022*, volume 13507 of *LNCS*, pages 719–749. Springer, 2022.

Sen11. Nicolas Sendrier. Decoding one out of many. In *PQCrypto*, volume 7071 of *LNCS*, pages 51–67. Springer, 2011.

Ste88. Jacques Stern. A method for finding codewords of small weight. In *Coding Theory and Applications*, volume 388 of *LNCS*, pages 106–113. Springer, 1988.

SZ13. Thomas Schneider and Michael Zohner. GMW vs. yao? efficient secure two-party computation with low depth circuits. In *Financial Cryptography*, volume 7859 of *LNCS*, pages 275–292. Springer, 2013.

TS16. Rodolfo Canto Torres and Nicolas Sendrier. Analysis of information set decoding for a sub-linear error weight. In *PQCrypto*, volume 9606 of *LNCS*, pages 144–161. Springer, 2016.

WRK17. Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Authenticated garbling and efficient maliciously secure two-party computation. In *CCS*, pages 21–37. ACM, 2017.

Yao86. Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS 1986*, pages 162–167. IEEE Computer Society, 1986.

YWL+20. Kang Yang, Chenkai Weng, Xiao Lan, Jiang Zhang, and Xiao Wang. Ferret: Fast extension for correlated OT with small communication. In *CCS '20*, pages 1607–1626. ACM, 2020.

ZGY+24. Wenhao Zhang, Xiaojie Guo, Kang Yang, Ruiyu Zhu, Yu Yu, and Xiao Wang. Efficient actively secure DPF and ram-based 2pc with one-bit leakage. In *IEEE Symposium on Security and Privacy, SP 2024*, pages 561–577. IEEE, 2024.

Supplementary Material

A More Preliminaries

Programmable PCG Programmability is a crucial property that allows for extending 2-party correlations from PCG to multi-party correlations. To define programmability of PCG, suppose $\mathcal{C}^N$ is a simple bilinear 2-party correlation (specified by a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ for some groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$).

Definition A.1 (Programmability). *We say a PCG* $\mathsf{PCG} = (\mathsf{PCG.Gen}, \mathsf{PCG.Expand})$ *for a simple bilinear* 2*-party correlation* $\mathcal{C}$ *satisfies programmability, if* $\mathsf{PCG.Gen}(1^\lambda)$ *takes additional random inputs* $\rho_0, \rho_1 \in \{0,1\}^\ell$*, for a* $\mathsf{poly}(\lambda)$ *size* ℓ*, such that:*

- ***Programmability.*** *There exist public efficiently computable functions* $\psi_0 : \{0,1\}^\ell \to \mathbb{G}_1^N$*,* $\psi_1 : \{0,1\}^\ell \to \mathbb{G}_2^N$ *such that*

$$\Pr\left[\begin{array}{l}\rho_0, \rho_1 \xleftarrow{\$} \{0,1\}^\lambda, (\mathbf{k}_0, \mathbf{k}_1) \xleftarrow{\$} \mathsf{PCG.Gen}(1^\lambda, \rho_0, \rho_1) \\ (R_0, S_0) \leftarrow \mathsf{PCG.Expand}(0, \mathbf{k}_0) \\ (R_1, S_1) \leftarrow \mathsf{PCG.Expand}(1, \mathbf{k}_1)\end{array} : \begin{array}{l} R_0 = \psi_0(\rho_0) \\ R_1 = \psi_1(\rho_1)\end{array}\right] \geq 1 - \mathsf{negl}(\lambda),$$

- ***Programmable security.*** *The following pair of distributions are computationally indistinguishable:*

$$\left\{(\mathbf{k}_1, (\rho_0, \rho_1)) \mid \rho_0, \rho_1 \xleftarrow{\$} \{0,1\}^\ell, (\mathbf{k}_0, \mathbf{k}_1) \xleftarrow{\$} \mathsf{PCG.Gen}(1^\lambda, \rho_0, \rho_1)\right\} \text{ and}$$
$$\left\{(\mathbf{k}_1, (\rho_0, \rho_1)) \mid \rho_0, \rho_1, \tilde{\rho}_0 \xleftarrow{\$} \{0,1\}^\ell, (\mathbf{k}_0, \mathbf{k}_1) \xleftarrow{\$} \mathsf{PCG.Gen}(1^\lambda, \tilde{\rho}_0, \rho_1)\right\}$$

  as well as

$$\left\{(\mathbf{k}_0, (\rho_0, \rho_1)) \mid \rho_0, \rho_1 \xleftarrow{\$} \{0,1\}^\ell, (\mathbf{k}_0, \mathbf{k}_1) \xleftarrow{\$} \mathsf{PCG.Gen}(1^\lambda, \rho_0, \rho_1)\right\} \text{ and}$$
$$\left\{(\mathbf{k}_0, (\rho_0, \rho_1)) \mid \rho_0, \rho_1, \tilde{\rho}_1 \xleftarrow{\$} \{0,1\}^\ell, (\mathbf{k}_0, \mathbf{k}_1) \xleftarrow{\$} \mathsf{PCG.Gen}(1^\lambda, \rho_0, \tilde{\rho}_1)\right\}$$

Ring-LPN Assumption. We define the Ring-LPN assumption as follows, which can be viewed as a univariate variant of QA-SD assumption. We refer security analysis of such Ring-LPN instantiation to [BCG+20, BCCD23, LWYY24].

Definition A.2 (Search Ring-LPN). *Let* $\mathcal{R} = \mathbb{F}_q[\mathtt{X}]/(\mathtt{X}^N - 1)$*, where* $N \mid q-1$*. Let* $c \geq 2$ *be some constant integer called the compression factor. Let* $\boldsymbol{a} = (1, a_1, \ldots, a_{c-1})$*, where* $a_i \xleftarrow{\$} \mathcal{R}$*,* $i \in [1, c-1]$*. Let* $e_0, e_1, \ldots, e_{c-1}$ *be random* t*-sparse elements of* $\mathcal{R}$*. Given access to a pair of the form* $(\boldsymbol{a}, \langle \boldsymbol{a}, \boldsymbol{e} \rangle)$*, the goal is to recover* $\boldsymbol{e}$*.*

Definition A.3 (Decisional Ring-LPN). *Let* $\mathcal{R} = \mathbb{F}_q[\mathtt{X}]/(\mathtt{X}^N - 1)$*, where* $N \mid q-1$*. Let* $c \geq 2$ *be some constant integer called the compression factor. The*

goal is to distinguish the following two distributions:

$$\mathcal{D}_0 : \{(a_1, \ldots, a_{c-1}, u)\}, \text{ where } a_i, u \xleftarrow{\$} \mathcal{R}, i \in [1, c-1]$$

$$\mathcal{D}_1 : \{(a_1, \ldots, a_{c-1}, \langle \boldsymbol{a}, \boldsymbol{e} \rangle + e_0)\}, \text{ where } a_i \xleftarrow{\$} \mathcal{R}, e_0, e_i \text{ are random } t\text{-sparse elements of } \mathcal{R}, i \in [1, c-1].$$

For simplicity, we consider the compression factor $c = 2$ as well.

B Deferred Proofs for Ring Isomorphisms and Traces Functions

Proof (Proof of Lemma 4.1). Let $\alpha_1 \ldots \alpha_N$ be N distinct roots of $\mathtt{X}^N = 1$ over $\mathbb{F}_q$. Then define

$$\begin{aligned} \phi : \mathcal{R} &\to \mathbb{F}_q^N \\ f &\mapsto (f(\alpha_1) \ldots f(\alpha_n)) \in \mathbb{F}_q^N. \end{aligned}$$

We first prove ϕ is a bijection from $\mathcal{R}$ to $\mathbb{F}_q^N$. Assume $f = \sum_{i=0}^{N-1} f_i \mathtt{X}^i$. Then the map ϕ is a linear transformation associated with $\alpha := (\alpha_1 \ldots \alpha_n)$. I.e., ϕ maps $(f_0 \ldots f_{N-1})$ to

$$(f_0 \ldots f_{N-1}) \cdot M_\alpha \text{ where } M_\alpha := \begin{pmatrix} \alpha_1^0 & \ldots & \alpha_n^0 \\ \vdots & \vdots & \vdots \\ \alpha_1^{N-1} & \ldots & \alpha_n^{N-1} \end{pmatrix}.$$

Then ϕ is a bijection from $\mathcal{R}$ to $\mathbb{F}_q^N$ as the matrix M_α is nonsingular.

Obliviously, $1_{\mathcal{R}} = 1 \in \mathcal{R}$ and $1_{\mathbb{F}_q^n} = \mathbf{1} \in \mathbb{F}_q^N$. Additionally, $\phi(1) = (1 \ldots 1)$. Thus, ϕ indeed maps $1_{\mathcal{R}}$ to $1_{\mathbb{F}_q^n}$.

Next, we prove the additive homomorphism. For any $f_1(\mathtt{X}), f_2(\mathtt{X}) \in \mathcal{R}$ with $f_1 = (f_{1,0} \ldots f_{1,N-1})$ and $f_2 = (f_{2,0} \ldots f_{2,N-1})$, we have

$$\phi\left(f_1(\mathtt{X}) + f_2(\mathtt{X})\right) = \phi\left(\sum_{i=0}^{N-1} (f_{1,i} + f_{2,i}) \mathtt{X}^i\right) = (f_{1,0} + f_{2,0} \ldots f_{1,N-1} + f_{2,N-1}) \cdot M_\alpha$$

and

$$\phi(f_1(\mathtt{X})) + \phi(f_2(\mathtt{X})) = (f_{1,0} \ldots f_{1,N-1}) \cdot M_\alpha + (f_{2,0} \ldots f_{2,N-1}) \cdot M_\alpha.$$

Thus, $\phi(f_1(\mathtt{X}) + f_2(\mathtt{X})) = \phi(f_1(\mathtt{X})) + \phi(f_2(\mathtt{X}))$. Therefore, ϕ preserves the additive group structure.

Next, we prove the multiplicative homomorphism. For any $f_1(\mathtt{X}), f_2(\mathtt{X}) \in \mathcal{R}$, the k-th entry of $\phi(f_1(\mathtt{X})) * \phi(f_2(\mathtt{X}))$ is

$$f_1(\alpha_k) \cdot f_2(\alpha_k) = \left(\sum_i f_{1,i} \alpha_k^i\right) \cdot \left(\sum_i f_{2,i} \alpha_k^i\right) = \sum_j \left(\sum_i f_{1,i} \cdot f_{2,j-i \pmod N}\right) \alpha_k^j.$$

Let $g(\mathtt{X}) := f_1(\mathtt{X}) \cdot f_2(\mathtt{X}) \mod (\mathtt{X}^N - 1)$. Then the j-th coefficient is $g_j := \sum_i f_{1,i} \cdot f_{2,j-i \pmod N}$. Thus

$$g(\alpha_k) = \sum_j g_j \cdot \alpha_k^j = \sum_j \left(\sum_i f_{1,i} \cdot f_{2,j-i \pmod N} \alpha_k^j \right).$$

It is $\phi(f_1(\mathtt{X}) \cdot f_2(\mathtt{X})) = \phi(f_1(\mathtt{X})) * \phi(f_2(\mathtt{X}))$.

Hence, the isomorphism ϕ respects addition, multiplication and the identity element. □

Proof (Proof of Lemma 4.2). Let $(\alpha_1 \dots \alpha_d)$ be d distinct root of $\mathtt{X}^d = 1$ over $\mathbb{F}_q$. Then define

$$\begin{aligned} \phi : \mathcal{R} &\to \mathbb{F}_q^{dn} \\ f &\mapsto (f(\alpha_{i_1}) \dots f(\alpha_{i_n}))_{i_1 \in [d] \dots i_t \in [d]} \in \mathbb{F}_q^{dn}. \end{aligned}$$

Assume

$$f = \sum_i f_i \cdot \mathtt{X}_1^{d_{i_1}} \cdots \mathtt{X}_n^{d_{i_n}}.$$

Denote $M := \otimes^n M_\alpha \in \mathbb{F}_q^{dn \times dn}$ as the n-th Kronecker product of M_α. Specifically, the matrix M can be constructed via an inductive method. Assume $M' \in \mathbb{F}_q^{d(n-1) \times d(n-1)}$ corresponds to the matrix for $n-1$ variables. Then the matrix for n variables is

$$\begin{pmatrix} \alpha_1^0 \cdot M' & \alpha_2^0 \cdot M' & \dots & \alpha_d^0 \cdot M' \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ \alpha_1^{d-1} \cdot M' & \alpha_2^{d-1} \cdot M' & \dots & \alpha_d^{d-1} \cdot M' \end{pmatrix} \in \mathbb{F}_q^{dn \times dn},$$

which is exactly $M_\alpha \otimes M'$.

Then ϕ maps $f = (f_0 \dots f_{dn-1})$ to $(f_0 \dots f_{dn-1}) \cdot M$. The matrix M is nonsingular as $\det(M) = \det(M_\alpha)^{nd^{n-1}} \neq 0$. Hence, ϕ is bijective.

Note that ϕ maps $1_{\mathcal{R}} = 1 \in \mathcal{R}$ to the $\mathbf{1} = 1_{\mathbb{F}_q^{dn}} \in \mathbb{F}_q^{dn}$. Obliviously, ϕ preserves the additive group structure.

Next, we prove the multiplicative homomorphism. Assume there is a canonical order for $\mathtt{X}_1^{d_1} \cdot \mathtt{X}_2^{d_2} \cdots \mathtt{X}_n^{d_n}$ and there exists a map ψ from $\mathbb{Z}$ to $\{\mathtt{X}_1^{d_1} \cdot \mathtt{X}_2^{d_2} \cdots \mathtt{X}_n^{d_n}\}_{d_i \in [d]}$. For simplicity, denote $\mathtt{X}^{[i]}$ as $\mathtt{X}^{[i]} := \mathtt{X}_1^{i_1} \cdots \mathtt{X}_n^{i_n}$ and thus define $i \oplus j := [\psi(\mathtt{X}^{[i]} \cdot \mathtt{X}^{[j]})]$. Similarly, denote $\alpha_{[k]}$ as $\alpha_{[k]} := (\alpha_{k_1}, \dots, \alpha_{k_n})$. For any $f_1(\mathtt{X}), f_2(\mathtt{X}) \in \mathcal{R}$, the k-th entry of $\phi(f_1(\mathtt{X})) * \phi(f_2(\mathtt{X}))$ is

$$f_1(\alpha_{[k]}) \cdot f_2(\alpha_{[k]}) = \left(\sum_i f_{1i} \alpha_{[k]}^{[i]} \right) \cdot \left(\sum_i f_{2,i} \alpha_{[k]}^{[i]} \right) = \sum_j \left(\sum_i f_{1,i} \cdot f_{2,j \ominus i} \right) \alpha_{[k]}^{[j]}.$$

Assume $g(\mathtt{X}) := f_1(\mathtt{X}) \cdot f_2(\mathtt{X})$. Then the j-th coefficient of $g(\mathtt{X})$ is $g_j := \sum_i f_{1,i} f_{2,j \ominus i}$. Thus,

$$g(\alpha_{[k]}) = \sum_j g_j \cdot \alpha_{[k]}^{[j]} = \sum_j \left(\sum_i f_{1,i} f_{2,j \ominus i} \cdot \alpha_{[k]}^{[j]} \right).$$

It is $\phi(f_1(\mathtt{X})) * \phi(f_2(\mathtt{X})) = \phi(f_1(\mathtt{X}) \cdot f_2(\mathtt{X}))$.

Hence, the isomorphism ϕ respects addition, multiplication and the identity element. □

An alternative method to prove this is the mathematical induction via viewing

$$\mathcal{R} = \mathbb{F}_q[\mathtt{X}_1 \dots \mathtt{X}_n]/(\mathtt{X}_1^d - 1, \dots, \mathtt{X}_n^d - 1) = (\mathbb{F}_q[\mathtt{X}_1, \dots, \mathtt{X}_{n-1}]/(\mathtt{X}_1^d - 1, \dots, \mathtt{X}_{n-1}^d - 1))[\mathtt{X}_t]/(\mathtt{X}_n^d - 1).$$

Proof (Proof of Lemma 4.3).

1. For arbitrary $\alpha \in \mathcal{R}$, $p \cdot \alpha = 0$. Thus, $\mathcal{R}$ is of characteristic p.
2. We prove it by induction. From the binomial expansion,

$$(a+b)^p = a^p + \binom{p}{1} a^{p-1}b + \cdots + \binom{p}{p-1} ab^{p-1} + b^p = a^p + b^p.$$

   Induction j, $(a+b)^{p^j} = (a^{p^{j-1}} + b^{p^{j-1}})^p = a^{p^j} + b^{p^j}$.
3. Assume a is of t-sparse and there exist t nonzero coefficients $(a_{i_1} \dots a_{i_t}) \in \mathbb{F}_{p^k}^t$ such that $a := \sum_{j \in [1,t]} a_{i_j} \mathtt{X}^{d_{i_j}}$. Then $a^p = (\sum_{j \in [1,t]} a_{i_j} \mathtt{X}^{d_{i_j}})^p = \sum_{j \in [1,t]} a_{i_j}^p \mathtt{X}^{p \cdot d_{i_j}}$ and thus a^p is still t-sparse. Inductively, a^{p^j} is t-sparse.
4. For univariate ring $\mathbb{F}_{p^k}[\mathtt{X}]/(\mathtt{X}^d - 1)$, it holds that $\mathtt{X}^{p^k} = \mathtt{X} \pmod{\mathtt{X}^d - 1}$ as $N \mid (p^k - 1)$. For multivariate ring $\mathcal{R} = \mathbb{F}_{p^k}[\mathtt{X}_1, \dots, \mathtt{X}_n]/(\mathtt{X}_1^d - 1, \dots, \mathtt{X}_n^d - 1)$, it holds that $\mathtt{X}_i^{p^k} = \mathtt{X}_i \pmod{\mathtt{X}_i^d - 1}$ as $d \mid (p^k - 1)$. Assume $a = \sum_{j \in [0, N-1]} a_j \mathtt{X}^j$. Then

$$a^{p^k} = (\sum_{j \in [0,N-1]} a_j \mathtt{X}^j)^{p^k} = \sum_{j \in [0,N-1]} a_j^{p^k} (\mathtt{X}^j)^{p^k} = \sum_j a_j \mathtt{X}^j = a.$$

□

An alternative method to prove this is the mathematical induction via viewing

$$\mathcal{R} = \mathbb{F}_q[\mathtt{X}_1 \dots \mathtt{X}_t]/(\mathtt{X}_1^n - 1, \dots, \mathtt{X}_t^n - 1) = (\mathbb{F}_q[\mathtt{X}_1, \dots, \mathtt{X}_{t-1}]/(\mathtt{X}_1^n - 1, \dots, \mathtt{X}_{t-1}^n - 1))[\mathtt{X}_t]/(\mathtt{X}_t^n - 1).$$

Proof (Proof of Theorem 4.1). We directly prove $\phi(\mathsf{Tr}(f)) = \mathsf{Tr}(\phi(f)) \in \mathbb{F}_p^N$. Assume $\phi(f) = \mathbf{v} \in \mathbb{F}_{p^k}^N$. Then

$$\mathsf{Tr}(\mathbf{v}) = \sum_{i=0}^{k-1} \mathbf{v}^{p^i} = \sum_{i=0}^{k-1} (\phi(f))^{p^i} = \sum_{i=0}^{k-1} \phi(f^{p^i}) = \phi(\sum_{i=0}^{k-1} f^{p^i}) = \phi(\mathsf{Tr}(f)),$$

where the third equality and the fourth equality follow from the ring isomorphism. □

Proof (Proof of Theorem 4.2).

1. For $f_1, f_2 \in \mathcal{R}$, from the properties of $\mathcal{R}$, we have $(f_1 + f_2)^p = f_1^p + f_2^p$.

$$\mathsf{Tr}(f_1 + f_2) = \sum_{j=0}^{k-1} (f_1 + f_2)^{p^j} = \sum_{j=0}^{k-1} (f_1^{p^j} + f_2^{p^j}) = \mathsf{Tr}(f_1) + \mathsf{Tr}(f_2).$$

2. $\mathsf{Tr}(\alpha \cdot f) = \sum_{j=0}^{k-1} (\alpha \cdot f)^{p^j} = \sum_{j=0}^{k-1} \alpha \cdot f^{p^j} = \alpha \cdot \mathsf{Tr}(f)$.
3. From the properties of $\mathcal{R}$, we have $f^{p^k} = f$. Then $\mathsf{Tr}(f^p) = \sum_{j=0}^{k-1} (f^p)^{p^j} = f^p + f^{p^2} + \cdots + f^{p^k} = \mathsf{Tr}(f)$.
4. It suffices to prove that for $f_1, f_2 \in \mathcal{R}$, $\phi(\mathsf{Tr}(f_1) \cdot \mathsf{Tr}(f_2)) = \phi(\mathsf{Tr}(\mathsf{Tr}(f_1) \cdot f_2))$ as ϕ is a bijection and applying ϕ^{-1} leads to the desired result. Then,

   $$\phi(\mathsf{Tr}(\mathsf{Tr}(f_1)\cdot f_2)) = \mathsf{Tr}(\phi(\mathsf{Tr}(f_1)\cdot f_2)) = \mathsf{Tr}(\phi(\mathsf{Tr}(f_1))*\phi(f_2)) = \phi(\mathsf{Tr}(f_1))*\mathsf{Tr}(\phi(f_2))$$

   where the first equality follows from Theorem 4.1, the second equality follows the ring isomorphism, and the third equality follows the fact that trace is an $\mathbb{F}_p$-linear map and $\phi(\mathsf{Tr}(f_1))$ is over $\mathbb{F}_p$. Moreover,

   $$\phi(\mathsf{Tr}(f_1)) * \mathsf{Tr}(\phi(f_2)) = \phi(\mathsf{Tr}(f_1)) * \phi(\mathsf{Tr}(f_2)) = \phi(\mathsf{Tr}(f_1) \cdot \mathsf{Tr}(f_2)),$$

   as desired. Hence $\mathsf{Tr}(\mathsf{Tr}(f_1) \cdot f_2) = \mathsf{Tr}(f_1) \cdot \mathsf{Tr}(f_2)$. Since $\mathcal{R}$ is commutative, we have $\mathsf{Tr}(f_1 \cdot \mathsf{Tr}(f_2)) = \mathsf{Tr}(f_1) \cdot \mathsf{Tr}(f_2)$.

□

To obtain $\mathbb{F}_p$ elements, one can compute the trace function on $\mathcal{R}$ elements and then apply the function ϕ.

Proof (Proof of Theorem 4.3). Note that we have a bijection $\phi : \mathcal{R} \to \mathbb{F}_{p^k}^N$, it suffices to prove that for a random $x \in \mathbb{F}_{p^k}$, $(\mathsf{Tr}(\xi^0 x), \mathsf{Tr}(\xi^1 x) \dots \mathsf{Tr}(\xi^{k-1} x)) \in \mathbb{F}_p^k$ is a linear transformation of $(x_1 \dots x_k) \in \mathbb{F}_p^k$ where $(x_1 \dots x_k) \in \mathbb{F}_p^k$ is the coefficient of x with respect to a given basis $(b_1 \dots b_k)$ of $\mathbb{F}_{p^k}$ over $\mathbb{F}_p$.

Lemma B.1. *Given $(b_1 \dots b_k) \in \mathbb{F}_{p^k}^k$ a basis of $\mathbb{F}_{p^k}$ over $\mathbb{F}_p$ and an arbitrary $x = \sum_{i\in[k]} x_i \cdot b_i \in \mathbb{F}_{p^k}$ with $x_i \in \mathbb{F}_p$, $(\mathsf{Tr}(x \cdot b_1) \dots \mathsf{Tr}(x \cdot b_k))$ is a linear transformation of $(x_1 \dots x_k)$.*

Proof. Before the proof we first define the discriminant matrix of a tuple $(b_1 \dots b_k)$.

Definition B.1 (Discriminant Matrix of a Tuple). *The discriminant matrix of a tuple $(b_1 \dots b_k) \in \mathbb{F}_{p^k}^k$ is defined as*

$$\mathsf{Disc}(b_1 \dots b_k) := \begin{bmatrix} \mathsf{Tr}(b_1 b_1) & \dots & \mathsf{Tr}(b_1 b_k) \\ \mathsf{Tr}(b_2 b_1) & \dots & \mathsf{Tr}(b_2 b_k) \\ \vdots & \vdots & \vdots \\ \mathsf{Tr}(b_k b_1) & \dots & \mathsf{Tr}(b_k b_k) \end{bmatrix}.$$

Then the discriminant of $(b_1 \dots b_k) \in \mathbb{F}_{p^k}^k$ is defined as $\Delta(b_1 \dots b_k) = \det(\mathsf{Disc}(b_1 \dots b_k))$.

By [LN97, Theorem 2.37], if $(b_1 \dots b_k)$ is a basis of $\mathbb{F}_{p^k}$ over $\mathbb{F}_p$, then $\det(\mathsf{Disc}(b_1 \dots b_k)) \neq 0$. Then $\mathsf{Tr}(x \cdot b_j) = \sum_{i\in[k]} x_i \cdot \mathsf{Tr}(b_i b_j) = (x_1 \dots x_k) \cdot (\mathsf{Tr}(b_1 b_j) \dots \mathsf{Tr}(b_k b_j))^{\mathrm{T}}$, for all $j \in [1, k]$. Thus,

$$(\mathsf{Tr}(x\cdot b_1) \dots \mathsf{Tr}(x\cdot b_k)) = (x_1 \dots x_k) \begin{bmatrix} \mathsf{Tr}(b_1 b_1) & \dots & \mathsf{Tr}(b_1 b_k) \\ \vdots & \vdots & \vdots \\ \mathsf{Tr}(b_k b_1) & \dots & \mathsf{Tr}(b_k b_k) \end{bmatrix} = (x_1 \dots x_k)\mathsf{Disc}(b_1 \dots b_k).$$

Therefore, the trace function $(\mathsf{Tr}(x \cdot b_1) \dots \mathsf{Tr}(x \cdot b_k))$ *of* $x = \sum_{i \in [k]} x_i \cdot b_i$ *induces a linear transformation from* $\mathbb{F}_p^k$ *to* $\mathbb{F}_p^k$. □

Hence, a uniform x in $\mathbb{F}_{p^k}$ induces a uniform tuple $(\mathsf{Tr}(x \cdot b_1) \dots \mathsf{Tr}(x \cdot b_k))$ over $\mathbb{F}_p^k$. In particular, $(1, \xi^1 \dots \xi^{k-1})$ and $(\xi, \xi^{p^1} \dots \xi^{p^{k-1}})$ are two bases of $\mathbb{F}_{p^k}$ over $\mathbb{F}_p$. Given a uniformly random $x \in \mathbb{F}_{p^k}$, $(\mathsf{Tr}(x \cdot \xi^0) \dots \mathsf{Tr}(x \cdot \xi^{k-1}))$ is uniformly random over $\mathbb{F}_p^k$. This completes the proof. □

Proof (Proof of Theorem 4.4). Since $k \mid \eta$, there exists $\zeta \in \mathbb{F}_{p^\eta}$ such that $\mathbb{F}_{p^\eta} = \mathbb{F}_{p^k}[\zeta]$, so that the natural embedding $\mathbb{F}_{p^k} \hookrightarrow \mathbb{F}_{p^\eta}$ implies a natural embedding $\mathcal{R}_k \hookrightarrow \mathcal{R}_\eta$. In addition, it follows that $d \mid p^\eta - 1$ as well, since $d \mid p^k - 1$. Together by Lemma 4.2, we complete the proof. □

In particular, let $p = 2, k = 2, d = 3$, then Theorem 4.4 actually holds for any even η. Namely, an isomorphism $\phi_2 : \mathcal{R}_2 = \mathbb{F}_4[\mathtt{X}_1, \dots, \mathtt{X}_n]/(\mathtt{X}_1^3 - 1, \dots, \mathtt{X}_n^3 - 1) \to \mathbb{F}_4^{3^n}$ determines an isomorphism $\phi_\eta : \mathcal{R}_\eta = \mathbb{F}_{2^\eta}[\mathtt{X}_1, \dots, \mathtt{X}_n]/(\mathtt{X}_1^3 - 1, \dots, \mathtt{X}_n^3 - 1) \to \mathbb{F}_{2^\eta}^{3^n}$.

C Distributed DPF Setup

In this section, we review the existing distributed DPF setup results from [BCG+20, Section 5], which was used to estimate the cost of our PCG setup protocols in Section 8. For semi-honest adversaries, the functionality for generating DPF keys is given in $\mathcal{F}_{\mathsf{DPF}}$.

Functionality 12: $\mathcal{F}_{\mathsf{DPF}}$

Parameters: Distributed point function $\mathsf{DPF} := (\mathsf{DPF.Gen}, \mathsf{DPF.Eval})$ with domain size N range $\mathbb{F} = \mathbb{F}_q$, where $N, q, \in \mathbb{N}$. Let λ be the security parameter.
Functionality:
DPF: On input $[\alpha]^{\{0,1\}^{\lceil \log N \rceil}}$ and $[\beta]^{\mathbb{F}}$:

1. Sample $(\mathsf{k}_0^{\mathsf{DPF}}, \mathsf{k}_1^{\mathsf{DPF}}) \leftarrow \mathsf{DPF.Gen}(1^\lambda, \alpha, \beta)$.
2. Output $\mathsf{k}_\sigma^{\mathsf{DPF}}$ to party P_σ for $\sigma \in \{0, 1\}$.

Lemma C.1 (Semi-honest Distributed DPF Key Generation). *[BCG+20, Section 5.2] For a point function with domain size N and range $\mathbb{F}_q$, there exists a protocol realizing the semi-honest DPF key generation functionality $\mathcal{F}_{\mathsf{DPF}}$ with the following complexity:*

- ***Correlated randomness**: 2 multiplication triples over $\mathbb{F}_q$ and $2 \log N$ number λ string OTs.*
- ***Computation complexity**: dominated by $2N$ PRG calls.*
- ***Communication complexity**: $(2\lambda + 3) \log N + 5 \log q$ bits.*

Next, we show a recent result for the semi-honest DPF construction.

Lemma C.2. *[GYW+23, Adapted from Theorem 5] Given a circular correlation robust(CCR) function* $\mathsf{H} : \mathbb{F}_{2^\lambda} \to \mathbb{F}_{2^\lambda}$*, function* $\mathsf{Convert}_{\mathcal{R}} : \mathbb{F}_{2^{\lambda-1}} \to \mathcal{R}$*, and keyed hash function* $\mathsf{H}_S(x) := \mathsf{H}(S \oplus x)$ *with some key* $S \leftarrow \mathbb{F}_{2^\lambda}$*, there exists a protocol* Π_{DPF} *realizing functionality* $\mathcal{F}_{\mathsf{DPF}}$ *against semi-honest adversaries in the* $(\mathcal{F}_{\mathsf{COT}}, \mathcal{F}_{\mathsf{Rand}})$*-hybrid model. Here is a complexity summary.*

- Π_{DPF} *uses* n *COT tuples each party and one* $\mathcal{F}_{\mathsf{Rand}}$ *call.*
- *For* t *concurrent generation, the communication complexity is* $t(n+1)(\lambda+1)+\lambda+t\log|\mathcal{R}|$*. The amortized communication complexity is* $(n+1)(\lambda+1)+\frac{\lambda}{t}+\log|\mathcal{R}|$*.*
- *The KeyGen complexity:* $1.5N$ *random permutation(RP) calls.*
- *Round complexity:* $n+3$*.*
- *Single point evaluation:* n *RP calls.*
- *Full-domain evaluation:* $1.5N$ *RP calls.*
- *Seed size:* $n\lambda+(\lambda+1)+\log|\mathcal{R}|$*.*

The function H *and random permutation can be instantiated from fixed-key AES-NI. Here are the details.*

- *A CCR function* H *can be constructed from a fixed-key block cipher. One CCR function call takes one block cipher invocation.*
- *The random permutation can be instantiated via fixed-key AES-NI.*
- *For* $\mathcal{R}$ *is very small, the* $\mathsf{Convert}_{\mathcal{R}}$ *can be instantiated from a PRG-free implementation.*

It is worth to mention there exists an efficient DPF scheme for ternary input point [BBC+24, Section 5.1].

For malicious adversaries, the functionality for generating DPF keys is given in $\mathcal{F}_{\mathsf{mal-DPF}}$. Note that the MAC sharing for (α, β) is implicitly included in the SPDZ sharing $\left([\![\alpha]\!]^{\{0,1\}^{\lceil \log N \rceil}}, [\![\beta]\!]^{\mathbb{F}_q}\right)$.

Functionality 13: $\mathcal{F}_{\mathsf{mal-DPF}}$

Parameters: Distributed point function $\mathsf{DPF} := (\mathsf{DPF.Gen}, \mathsf{DPF.Eval})$ with domain size N range $\mathbb{F}_q$, where $N, q, \in \mathbb{N}$. Let λ be the security parameter. Assume $\eta \in \mathbb{N}$ such that $q^\eta \geq 2^\lambda$. Denote $\Delta \in \mathbb{F}_{q^\eta}$ as the global key.

$\mathsf{mal\text{-}DPF}([\![\alpha]\!]^{\{0,1\}^{\lceil \log N \rceil}}, [\![\beta]\!]^{\mathbb{F}_q})$:

If both parties are honest,

1. If $\beta = 0$, output $\beta = 0$ to the two parties and abort.
2. Sample $\mathbf{y}_\sigma \xleftarrow{\$} (\mathbb{F}_q \times \mathbb{F}_{q^\eta})^N$ and set $y_{1-\sigma} \leftarrow (0 \dots 0, \beta \cdot (1, \Delta) \dots 0) - \mathbf{y}_\sigma \in (\mathbb{F}_q \times \mathbb{F}_{q^\eta})^N$, where $\beta \cdot (1, \Delta)$ is at the α-th position.
3. Output $\mathbf{y}_\sigma$ to P_σ for $\sigma \in \{0, 1\}$.

If P_σ is corrupted,

1. Wait for input $\mathbf{y}_\sigma \in (\mathbb{F}_q \times \mathbb{F}_{q^\eta})^N$ from the adversary ($\mathcal{A}$ determines the output shares).

2. Wait for a predicate $P : [0, N-1] \to \{0,1\}$ from the adversary ($\mathcal{A}$ queries a predicate on α). If $P(\alpha) = 0$, abort.
3. If $\beta = 0$, output $\beta = 0$ to the two parties and abort.
4. Set $\mathbf{y}_{1-\sigma} \leftarrow (0 \dots 0, \beta \cdot (1, \Delta) \dots 0) - \mathbf{y}_\sigma \in (\mathbb{F}_q \times \mathbb{F}_{q^\eta})^N$.
5. Output success to the adversary and $\mathbf{y}_{1-\sigma}$ to the honest party $P_{1-\sigma}$.

Note that $\mathcal{F}_{\mathsf{mal-DPF}}$ allows some leakage on the position value α because during the key generation a corrupted party is able to guess each bit of α learn some bit of α. Different from $\mathcal{F}_{\mathsf{DPF}}$, $\mathcal{F}_{\mathsf{mal-DPF}}$ outputs the additive shares of the DPF expanding results rather than the DPF keys.

Lemma C.3 (Malicious Distributed DPF Setup). *[BCG+20, Section 5.3] For a point function with domain size N and range $\mathbb{F}_q$, there exists a protocol realizing $\mathcal{F}_{\mathsf{mal-DPF}}$ against malicious adversaries, .i.e., outputting the sharing of a scaled unit vector as DPF expanding, with the following complexity:*

- ***Correlated randomness**: 3 authenticated multiplication triples over $\mathbb{F}_q$ and 2 length-4 VOLE over $\mathbb{F}_q$.*
- ***Computation complexity**: dominated by $2N$ PRG calls.*
- ***Communication complexity**: $(2\lambda + 3)\log N + 11 \log q$ bits.*

The efficiency of the protocol implementing $\mathcal{F}_{\mathsf{mal-DPF}}$ can be plausibly further improved by using the techniques of a concurrent work [ZGY+24].

Functionality 14: $\mathcal{F}_{\mathsf{2PC}}$

The functionality operates on elements of $\mathbb{F} = \mathbb{F}_{p^k}$ for $k \in \mathbb{N}$ and elements of $\mathbb{G} = \mathbb{Z}_d^n$, where $n \in \mathbb{N}$ and $d \mid p^k - 1$. Let $N = |\mathbb{G}| = d^n$. Each value stored by the functionality is associated with a unique identifier that is given to all parties. Let $[\![x]\!]^{\mathbb{F}}$ denote the identifier for a value $x \in \mathbb{F}$ and $[\![x]\!]^{\mathbb{G}}$ denote the identifier for an element $x \in \mathbb{G}$.

Input(P_σ, x): Receive a value $x \in \mathbb{F}$ or $x \in \mathbb{G}$ from party P_σ and store $[\![x]\!]^{\mathbb{F}}$ or $[\![x]\!]^{\mathbb{G}}$.
Add($[\![x]\!]^{\mathbb{G}}, [\![y]\!]^{\mathbb{G}}$): Compute $z = x + y \in \mathbb{G}$ and store $[\![z]\!]^{\mathbb{G}}$.
ToBinary($[\![x]\!]^{\mathbb{G}}$): View $x \in \mathbb{G}$ as $\widetilde{x} \in \{0,1\}^{\lceil \log N \rceil}$, and store $[\![\widetilde{x}]\!]^{\{0,1\}^{\lceil \log N \rceil}}$.
PowerP($[\![x]\!]^{\mathbb{F}}, p$): Compute and store $[\![x^p]\!]^{\mathbb{F}}$.
Mult($[\![x]\!]^{\mathbb{F}}, [\![y]\!]^{\mathbb{F}}$): Compute $z = x \cdot y \in \mathbb{F}$ and store $[\![z]\!]^{\mathbb{F}}$.
Output($[\![x]\!]^{N}$): Send the value $x \in [0, N)$ to all parties.
Output($[\![x]\!]^{\mathbb{F}}$): Send the value $x \in \mathbb{F}$ to all parties.

D PCG Setup Protocols from Ring-LPN

For completeness, we give the Ring-LPN analogues of our results in this section. Our results from Ring-LPN are obtained by extending those based on QA-SD assumptions. Generally speaking, constructions from Ring-LPN mainly have

two advantages, i) Ring-LPN assumptions have been studied for a long time, ii) mature FFT algorithms and implementations (for fields $\mathbb{F}_{p^k}$ with small p, e.g., $p = 2$, we can use additive FFT [LAH16]). W.l.o.g., we focus on the binary field in this section.

D.1 PCG for OLE from Ring-LPN

For completeness, we show a Boolean OLE construction.

Construction 15: $\mathsf{Cons}^{\mathbb{F}_2}_{\mathsf{OLE}}$

Parameters: Security parameter λ, noise weight $t = t(\lambda)$, $N \mid 2^k - 1$, $\mathcal{R} = \mathbb{F}_{2^k}[\mathtt{X}]/(\mathtt{X}^N - 1)$, $\xi \in \mathbb{F}_{2^k}$ such that $\mathbb{F}_{2^k} = \mathbb{F}(\xi)$. An FSS scheme $(\mathsf{SPFSS.Gen}, \mathsf{SPFSS.FullEval})$ for sums of t^2 point functions, with domain $\mathbb{Z}_N$ and range $\mathbb{F}_{2^k}$. Fix a basis $(\xi, \xi^{2^1}, \ldots, \xi^{2^{k-1}})$ of $\mathbb{F}_{2^k}$ over $\mathbb{F}_2$.
Public input: A uniformly random $a \in \mathcal{R}$.
Correlation: For $\ell \in [0, k-1]$, output $(\mathbf{X}_0^{(\ell)}, \mathbf{Z}_0^{(\ell)}) \in \mathbb{F}_2^{2N}$ and $(\mathbf{X}_1^{(\ell)}, \mathbf{Z}_1^{(\ell)}) \in \mathbb{F}_2^{2N}$ such that $\mathbf{X}_0^{(\ell)} * \mathbf{X}_1^{(\ell)} = \mathbf{Z}_0^{(\ell)} + \mathbf{Z}_1^{(\ell)}$.
Gen: On input 1^λ,

1. For $\sigma \in \{0,1\}$, $i \in \{0,1\}$, sample random vectors $\mathbf{A}_\sigma^i \leftarrow \mathbb{Z}_N^t$ and $\mathbf{b}_\sigma^i \leftarrow (\mathbb{F}_{2^k}^*)^t$.
2. Sample FSS keys according to Eq.(7). For $j \in [0, k-1]$,
   $(K_0^{0,j}, K_1^{0,j}) \leftarrow \mathsf{SPFSS.Gen}(1^\lambda, \mathbf{A}_0^0 \boxplus (\mathbf{A}_1^0)^{2^j}, \mathbf{b}_0^0 \otimes (\mathbf{b}_1^0)^{2^j})$
   $(K_0^{1,j}, K_1^{1,j}) \leftarrow \mathsf{SPFSS.Gen}(1^\lambda, \mathbf{A}_0^1 \boxplus (\mathbf{A}_1^1)^{2^j}, \mathbf{b}_0^1 \otimes (\mathbf{b}_0^1)^{2^j})$
   $(K_0^{2,j}, K_1^{2,j}) \leftarrow \mathsf{SPFSS.Gen}(1^\lambda, \mathbf{A}_0^0 \boxplus (\mathbf{A}_1^0)^{2^j}, \mathbf{b}_0^0 \otimes (\mathbf{b}_1^1)^{2^j})$
   $(K_0^{3,j}, K_1^{3,j}) \leftarrow \mathsf{SPFSS.Gen}(1^\lambda, \mathbf{A}_0^1 \boxplus (\mathbf{A}_1^1)^{2^j}, \mathbf{b}_0^1 \otimes (\mathbf{b}_1^1)^{2^j})$
3. Set $\mathsf{k}_\sigma := \big(\{K_\sigma^{i,j}\}_{i\in[0,3],j\in[0,k-1]}, \{\mathbf{A}_\sigma^i, \mathbf{b}_\sigma^i\}_{i\in\{0,1\}}\big)$
4. Output $(\mathsf{k}_0, \mathsf{k}_1)$.

Expand: On input $(\sigma, \mathsf{k}_\sigma)$,

1. Parse k_σ as $\big(\{K_\sigma^{i,j}\}_{i\in[0,3],j\in[0,k-1]}, \{\mathbf{A}_\sigma^i, \mathbf{b}_\sigma^i\}_{i\in\{0,1\}}\big)$.
2. Define elements of $\mathcal{R}$: For $i \in \{0,1\}$, $//x_\sigma := a \cdot e_\sigma^0 + e_\sigma^1$

$$e_\sigma^i := \sum_{j\in[1,t]} \mathbf{b}_\sigma^i[j] \cdot \vec{\mathtt{X}}^{\mathbf{A}_\sigma^i[j]}.$$

3. For $\ell \in [0, k-1]$, compute $\mathbf{X}_\sigma^{(\ell)} := \mathsf{Tr}\big(\xi^{2^\ell} \cdot (a \cdot e_\sigma^0 + e_\sigma^1)\big) \in \mathbb{F}_2^N$.
4. For $i \in [0,3]$, $j \in [1, k-1]$, compute $w_\sigma^{i,j} := \mathsf{SPFSS.FullEval}(\sigma, K_\sigma^{i,j})$ as $\mathcal{R}$ element.
5. For $\ell \in [1, k-1]$, according to Eq (8), compute

$$\mathbf{Z}_\sigma^{(\ell)} := \sum_{j\in[0,k-1]} \mathsf{Tr}\big(\xi^{2^\ell+2^{\ell+j}} \cdot (a^{2^j+1} w_\sigma^{0,j} + a^{2^j} w_\sigma^{1,j} + a w_\sigma^{2,j} + w_\sigma^{3,j})\big)$$

6. Output $\{\mathbf{X}_\sigma^{(\ell)}, \mathbf{Z}_\sigma^{(\ell)}\}_{\ell\in[0,k-1]}$.

Theorem D.1. *Assume a secure* FSS *scheme* SPFSS *for sums of point functions and* Ring-LPN$(\mathcal{R}, t)$ *is hard. Then there exists a PCG construction* $\mathsf{Cons}^{\mathbb{F}_2}_{\mathsf{OLE}}$ *that generates OLE correlations over* $\mathbb{F}_2$*. If the* SPFSS *is based on a* PRG : $\{0,1\}^\lambda \to \{0,1\}^{2\lambda+2}$ *via the* PRG*-based construction from [BGI16], for generating* kN *OLE correlations over* $\mathbb{F}_2$*, we have that:*

- *Each party has seed size at most:* $4kt^2\big(\log N(\lambda+2)+\lambda+k\big)+2t(\log N+k)$ *bits.*
- *The* **Expand** *procedure makes at most* $4kt^2N(1+\lceil k/\lambda\rceil)$ *PRG calls and* $O(k^2N\log N)$ *operations over* $\mathbb{F}_{2^k}$*.*

Remark D.1. The differences compared to Theorem 5.1 (from QA-SD) are that now $k = O(\log N)$ *and* $p = 2$*. Note here the optimizations from assuming regular noise are not applied, since* N *might be a prime, e.g. when* k *is a Mersenne prime.*

D.2 Authenticated Boolean Triple Constructions from Ring-LPN

We show a PCG construction for authenticated Boolean triples from Ring-LPN assumptions, which is analogue to $\mathsf{Cons}'^{\mathbb{F}_p}_{\mathsf{Auth-Triple}}$ (from QA-SD). For simplicity, the construction makes use of SPDZ sharings.

Construction 16: $\mathsf{Cons}'^{\mathbb{F}_2}_{\mathsf{Auth-Triple}}$

Parameters: Security parameter 1^λ, noise weight $t = t(\lambda)$, η satisfying $\eta \geq \lambda$ and $k \mid \eta$, $\mathcal{R}_k = \mathbb{F}_{2^k}[\mathtt{X}]/(\mathtt{X}^N - 1)$ where $N \mid 2^k - 1$, $\xi \in \mathbb{F}_{2^k}$ s.t. $\mathbb{F}_{2^k} = \mathbb{F}_2(\xi)$. Let $\mathbb{G} := \mathbb{Z}_N$ and $\mathbb{F} := \mathbb{F}_{2^k}$. An FSS scheme (SPFSS.Gen, SPFSS.FullEval) for sums of point functions, with domain $\mathbb{Z}_N$ and range $\mathbb{F}_{2^k}$ or $\mathbb{F}_{2^\eta}$. We abuse ϕ as a fixed isomorphism map from $\mathcal{R}_k$ to $\mathbb{F}_{2^k}^N$, and from $\mathcal{R}_\eta$ to $\mathbb{F}_{2^\eta}^N$. The global MAC key shares $\Delta_0, \Delta_1 \in \mathbb{F}_{2^\eta}$ are implicitly provided by authenticated multiplication triples.
Public input: A uniformly random $a \in \mathcal{R}_k$.
Correlation: For $\ell \in [0, k-1]$, output $(\llbracket \mathbf{X}^{(\ell)} \rrbracket, \llbracket \mathbf{Y}^{(\ell)} \rrbracket, \llbracket \mathbf{Z}^{(\ell)} \rrbracket)$, where $\mathbf{X}^{(\ell)}, \mathbf{Y}^{(\ell)}, \mathbf{Z}^{(\ell)} \in \mathbb{F}_2^N$ such that $\mathbf{X}^{(\ell)} * \mathbf{Y}^{(\ell)} = \mathbf{Z}^{(\ell)}$ and MAC key shares $\Delta_0, \Delta_1 \in \mathbb{F}_{2^\eta}$.
Gen: On input 1^λ,

1. For $\sigma \in \{0,1\}$, $i \in \{0,1\}$, sample random vectors $\mathbf{A}^{x,i}_\sigma, \mathbf{A}^{y,i}_\sigma \xleftarrow{\$} \mathbb{G}^t$ and $\mathbf{b}^{x,i}_\sigma, \mathbf{b}^{y,i}_\sigma \xleftarrow{\$} (\mathbb{F}^*)^t$ for the positions and values. Let $\mathbf{A}^{x,i}, \mathbf{A}^{y,i} \in \mathbb{G}^{2t}$ and $\mathbf{b}^{x,i}, \mathbf{b}^{y,i} \in (\mathbb{F}^*)^{2t}$ be the union of the corresponding positions and values. Define elements of $\mathcal{R}_k$:

$$e^{x,i} := \sum_{j=1}^{2t} \mathbf{b}^{x,i}[j] \cdot \mathtt{X}^{\mathbf{A}^{x,i}[j]}, x := a \cdot e^{x,0} + e^{x,1}$$

$$e^{y,i} := \sum_{j=1}^{2t} \mathbf{b}^{y,i}[j] \cdot \mathtt{X}^{\mathbf{A}^{y,i}[j]}, y := a \cdot e^{y,0} + e^{y,1}$$

2. Share the value $\mathbf{A}^{x,i}, \mathbf{A}^{y,i} \in \mathbb{G}^{2t}$ and $\mathbf{b}^{x,i}, \mathbf{b}^{y,i} \in (\mathbb{F}^*)^{2t}$ to obtain $[\![\mathbf{A}^{x,i}]\!], [\![\mathbf{A}^{y,i}]\!]$ and $[\![\mathbf{b}^{x,i}]\!], [\![\mathbf{b}^{y,i}]\!]$ via consuming SPDZ sharings.
3. Sample FSS keys for $\left[\!\left[\mathbf{X}^{(\ell)}\right]\!\right], \left[\!\left[\mathbf{Y}^{(\ell)}\right]\!\right]$ according to Eq.(9). For $i \in \{0,1\}$, $j \in [0, k-1]$,
   - $(K_0^{x,i,j}, K_1^{x,i,j}) \leftarrow \mathsf{SPFSS.Gen}(1^\lambda, \boxplus^{2^j} [\![\mathbf{A}^{x,i}]\!], \otimes^{2^j} [\![\mathbf{b}^{x,i}]\!])$
   - $(K_0^{y,i,j}, K_1^{y,i,j}) \leftarrow \mathsf{SPFSS.Gen}(1^\lambda, \boxplus^{2^j} [\![\mathbf{A}^{y,i}]\!], \otimes^{2^j} [\![\mathbf{b}^{y,i}]\!])$
4. Sample FSS keys for $\left[\!\left[\mathbf{Z}^{(\ell)}\right]\!\right]$ according to Eq. (10). For $i, j \in [0, k-1]$,

$$(K_0^{z,i,j,0}, K_1^{z,i,j,0}) \leftarrow \mathsf{SPFSS.Gen}(1^\lambda, [\![\mathbf{A}^{x,0}]\!]^{2^i} \boxplus [\![\mathbf{A}^{y,0}]\!]^{2^j}, [\![\mathbf{b}^{x,0}]\!]^{2^i} \otimes [\![\mathbf{b}^{y,0}]\!]^{2^j})$$

$$(K_0^{z,i,j,1}, K_1^{z,i,j,1}) \leftarrow \mathsf{SPFSS.Gen}(1^\lambda, [\![\mathbf{A}^{x,0}]\!]^{2^i} \boxplus [\![\mathbf{A}^{y,1}]\!]^{2^j}, [\![\mathbf{b}^{x,0}]\!]^{2^i} \otimes [\![\mathbf{b}^{y,1}]\!]^{2^j})$$

$$(K_0^{z,i,j,2}, K_1^{z,i,j,2}) \leftarrow \mathsf{SPFSS.Gen}(1^\lambda, [\![\mathbf{A}^{x,1}]\!]^{2^i} \boxplus [\![\mathbf{A}^{y,0}]\!]^{2^j}, [\![\mathbf{b}^{x,1}]\!]^{2^i} \otimes [\![\mathbf{b}^{y,0}]\!]^{2^j})$$

$$(K_0^{z,i,j,3}, K_1^{z,i,j,3}) \leftarrow \mathsf{SPFSS.Gen}(1^\lambda, [\![\mathbf{A}^{x,1}]\!]^{2^i} \boxplus [\![\mathbf{A}^{y,1}]\!]^{2^j}, [\![\mathbf{b}^{x,1}]\!]^{2^i} \otimes [\![\mathbf{b}^{y,1}]\!]^{2^j})$$

5. For $\sigma \in \{0,1\}$, let $\mathsf{k}_\sigma = \left(\left\{K_\sigma^{x,i,j}, K_\sigma^{y,i,j}\right\}_{j\in[0,k-1]}^{i\in\{0,1\}}, \left\{K_\sigma^{z,i,j,\kappa}\right\}_{i,j\in[0,k-1]}^{\kappa\in[0,3]}\right)$.
6. Output $(\mathsf{k}_0, \mathsf{k}_1)$.

Expand: On input $(\sigma, \mathsf{k}_\sigma)$:

1. Parse k_σ as $\left(\left\{K_\sigma^{x,i,j}, K_\sigma^{y,i,j}\right\}_{j\in[0,k-1]}^{i\in\{0,1\}}, \left\{K_\sigma^{z,i,j,\kappa}\right\}_{i,j\in[0,k-1]}^{\kappa\in[0,3]}\right)$.
2. For $i \in \{0,1\}$, $j \in [0, k-1]$, compute //View $u^{i,j}, v^{i,j}, w^{ij\kappa}$ as $\mathcal{R}_k$ elements.

$$\left[\!\left[u^{i,j}\right]\!\right]_\sigma \leftarrow \mathsf{SPFSS.FullEval}(\sigma, K_\sigma^{x,i,j}), \left[\!\left[v^{i,j}\right]\!\right]_\sigma \leftarrow \mathsf{SPFSS.FullEval}(\sigma, K_\sigma^{y,i,j})$$

3. For $i, j \in [0, k-1], \kappa \in [0,3]$, compute

$$\left[\!\left[w^{ij\kappa}\right]\!\right]_\sigma \leftarrow \mathsf{SPFSS.FullEval}(\sigma, K_\sigma^{z,i,j,\kappa}).$$

4. For $\ell \in [0, k-1]$, compute

$$\left[\!\left[X^{(\ell)}\right]\!\right]_\sigma \leftarrow \phi\left(\sum_{j\in[0,k-1]} \xi^{2^{\ell+j}} \cdot \left(a^{2^j} \cdot \left[\!\left[u^{0,j}\right]\!\right]_\sigma + \left[\!\left[u^{1,j}\right]\!\right]_\sigma\right)\right)$$

$$\left[\!\left[Y^{(\ell)}\right]\!\right]_\sigma \leftarrow \phi\left(\sum_{j\in[0,k-1]} \xi^{2^{\ell+j}} \cdot \left(a^{2^j} \cdot \left[\!\left[v^{0,j}\right]\!\right]_\sigma + \left[\!\left[v^{1,j}\right]\!\right]_\sigma\right)\right)$$

5. For $\ell \in [0, k-1]$, according to Eq.(10), compute

$$\left[\!\left[Z^{(\ell)}\right]\!\right]_\sigma \leftarrow \phi\Big(\sum_{i,j\in[0,k-1]} \xi^{2^\ell(2^i+2^j)} \big(a^{2^i+2^j} \cdot \left[\!\left[w^{z,i,j,0}\right]\!\right]_\sigma + a^{2^i} \cdot \left[\!\left[w^{z,i,j,1}\right]\!\right]_\sigma + a^{2^j} \cdot \left[\!\left[w^{z,i,j,2}\right]\!\right]_\sigma + \left[\!\left[w^{z,i,j,3}\right]\!\right]_\sigma\big)\Big)$$

6. For $\ell \in [0, k-1]$, output $\left(\left[\!\left[\mathbf{X}^{(\ell)}\right]\!\right]_\sigma, \left[\!\left[\mathbf{Y}^{(\ell)}\right]\!\right]_\sigma, \left[\!\left[\mathbf{Z}^{(\ell)}\right]\!\right]_\sigma\right)$ to P_σ, where $\mathbf{X}^{(\ell)}, \mathbf{Y}^{(\ell)}, \mathbf{Z}^{(\ell)} \in \mathbb{F}_2^N$.

Theorem D.2. *Assume a secure* FSS *scheme* SPFSS *for sums of point functions and* Ring-LPN$(\mathcal{R}_k, t)$ *is hard. Then there exists a PCG construction* $\mathsf{Cons}^{\mathbb{F}_2}_{\mathsf{Auth-Triple}}$ *that generates authenticated Boolean triples. If the* SPFSS *is based on a* PRG : $\{0,1\}^\lambda \to \{0,1\}^{2\lambda+2}$ *via the* PRG*-based construction from [BGI16], to produce* kN *authenticated Boolean triples, we have that:*

- *Each party has seed size at most:* $(16k^2t^2 + 8kt) \cdot ((\lambda+2)\log N + \lambda + (\eta + k))$ *bits.*
- *The* **Expand** *procedure makes at most* $(16k^2t^2 + 8kt) \cdot (1 + \left\lceil \frac{(k+\eta)}{\lambda+2} \right\rceil) \cdot N$ *PRG calls and* $O(k^3 N \log N)$ *operations over* $\mathbb{F}_{2^k} \times \mathbb{F}_{2^\eta}$.

D.3 Semi-honest Distributed Setup from Ring-LPN

In this section, we show a protocol $\Pi^{\mathbb{F}_2}_{\mathsf{OLE-Setup}}$ that realizes the seed generation functionality $\mathcal{F}^{\mathbb{F}_2}_{\mathsf{OLE-Setup}}$ for OLE correlations from Ring-LPN assumptions against a semi-honest adversary. Note that after the setup protocol, each party obtains succinct representations of the sparse error vector via (postion, value) pairs and succinct representations of the products of error vectors via DPFs.

Functionality 17: $\mathcal{F}^{\mathbb{F}_2}_{\mathsf{OLE-Setup}}$

Parameters: Security parameter 1^λ, $\mathsf{PCG}_{\mathsf{OLE}} := (\mathsf{PCG}_{\mathsf{OLE}}.\mathsf{Gen}, \mathsf{PCG}_{\mathsf{OLE}}.\mathsf{Expand})$ in line with Construction $\mathsf{Cons}^{\mathbb{F}_2}_{\mathsf{OLE}}$.
Functionality:

1. Sample $(\mathsf{k}_0, \mathsf{k}_1) \leftarrow \mathsf{PCG}_{\mathsf{OLE}}.\mathsf{Gen}(1^\lambda)$.
2. For $\sigma \in \{0,1\}$, output k_σ to P_σ.

Protocol 18: $\Pi^{\mathbb{F}_2}_{\mathsf{OLE-Setup}}$

Parameters: Security parameter 1^λ, length $N \mid 2^k - 1$, $\mathbb{F}_{2^k} = \mathbb{F}_p(\xi)$ and $\mathcal{R} = \mathbb{F}_{2^k}[\mathtt{X}]/(\mathtt{X}^N - 1)$. Let $\mathbb{G} = \mathbb{Z}_N$. Assume $\mathbb{F} := \mathbb{F}_{2^k}$. Let DPF be a distributed point function $\mathsf{DPF} := (\mathsf{DPF.Gen}, \mathsf{DPF.Eval})$ with domain size N and range $\mathbb{F}_{2^k}$. Assume each variable is shared via additive sharing, for instance $[x] = (x_0, x_1)$. Furthermore, it is given access to the functionalities $\mathcal{F}_{\mathsf{2PC}}$ and $\mathcal{F}_{\mathsf{DPF}}$.
Protocol:

1. P_σ samples random vectors $\mathbf{A}^0_\sigma, \mathbf{A}^1_\sigma \xleftarrow{\$} \mathbb{G}^t$ and $\mathbf{b}^0_\sigma, \mathbf{b}^1_\sigma \xleftarrow{\$} (\mathbb{F}^*_{2^k})^t$. We remark that each pair $(\mathbf{A}^i_\sigma, \mathbf{b}^i_\sigma)$ defines a t-sparse element in $\mathcal{R}$.
2. P_0 inputs the positions and the values. For $i \in [0,1], j \in [1,t]$,
$$\left[\mathbf{A}^i_0[j]\right] \leftarrow \mathsf{Input}(P_0, \mathbf{A}^i_0[j]), \quad \left[\mathbf{b}^i_0[j]\right] \leftarrow \mathsf{Input}(P_0, \mathbf{b}^i_0[j]).$$
3. P_1 compute the positions and the values and then input them. For $i \in [0,1], j \in [1,t], \ell \in [0, k-1]$, compute $2^\ell \cdot \mathbf{A}^i_1[j]$ and $(\mathbf{b}^i_1[j])^{2^\ell}$ iteratively. Then,
$$\left[2^\ell \cdot \mathbf{A}^i_1[j]\right] \leftarrow \mathsf{Input}(P_1, 2^\ell \cdot \mathbf{A}^i_1[j]), \quad \left[(\mathbf{b}^i_1[j])^{2^\ell}\right] \leftarrow \mathsf{Input}(P_1, (\mathbf{b}^i_1[j])^{2^\ell}).$$

4. Compute the cross sums of positions and products of values. For $\ell \in [0, k-1]$, $i, j \in [1, t]$,

$$\left[\alpha_0^{ij\ell}\right] \leftarrow \mathsf{Add}(\left[\mathbf{A}_0^0[i]\right], \left[2^\ell \cdot \mathbf{A}_1^0[j]\right]), \left[\beta_0^{ij\ell}\right] \leftarrow \mathsf{Mul}(\left[\mathbf{b}_0^0[i]\right], \left[(\mathbf{b}_1^0[j])^{2^\ell}\right])$$

$$\left[\alpha_1^{ij\ell}\right] \leftarrow \mathsf{Add}(\left[\mathbf{A}_0^1[i]\right], \left[2^\ell \cdot \mathbf{A}_1^0[j]\right]), \left[\beta_1^{ij\ell}\right] \leftarrow \mathsf{Mul}(\left[\mathbf{b}_0^1[i]\right], \left[(\mathbf{b}_1^0[j])^{2^\ell}\right])$$

$$\left[\alpha_2^{ij\ell}\right] \leftarrow \mathsf{Add}(\left[\mathbf{A}_0^0[i]\right], \left[2^\ell \cdot \mathbf{A}_1^1[j]\right]), \left[\beta_2^{ij\ell}\right] \leftarrow \mathsf{Mul}(\left[\mathbf{b}_0^0[i]\right], \left[(\mathbf{b}_1^1[j])^{2^\ell}\right])$$

$$\left[\alpha_3^{ij\ell}\right] \leftarrow \mathsf{Add}(\left[\mathbf{A}_0^1[i]\right], \left[2^\ell \cdot \mathbf{A}_1^1[j]\right]), \left[\beta_3^{ij\ell}\right] \leftarrow \mathsf{Mul}(\left[\mathbf{b}_0^1[i]\right], \left[(\mathbf{b}_1^1[j])^{2^\ell}\right])$$

5. Convert the position value over $\mathbb{G} = \mathbb{Z}_N$ to binary value over $\{0,1\}^{\lceil \log |\mathbb{G}| \rceil}$. For $\kappa \in [0, 3], i, j \in [1, t], \ell \in [0, k-1]$,

$$\left[\alpha_\kappa^{ij\ell}\right]^{\{0,1\}^{\lceil \log |\mathbb{G}| \rceil}} \leftarrow \mathsf{ToBinary}(\left[\alpha_\kappa^{ij\ell}\right]^{\mathbb{G}})$$

6. Sample the FSS key shares via calling $\mathcal{F}_{\mathsf{DPF}}$ with domain size N and range $\mathbb{F}_{2^k}$. For $\kappa \in [0, 3], i, j \in [1, t], \ell \in [0, k-1]$,

$$(K_0^{ij\ell\kappa}, K_1^{ij\ell\kappa}) \leftarrow \mathcal{F}_{\mathsf{DPF}}(\left[\alpha_k^{ij\ell}\right]^{\{0,1\}^{\lceil \log |\mathbb{G}| \rceil}}, \left[\beta_k^{ij\ell}\right]).$$

7. P_σ outputs $\mathsf{k}_\sigma := \left(\{K_\sigma^{ij\ell\kappa}\}_{i,j\in[1,t],\kappa\in[0,3]}^{\ell\in[0,k-1]}, \{\mathbf{A}_\sigma^i, \mathbf{b}_\sigma^i\}_{i\in\{0,1\}}\right)$.

Theorem D.3. *The protocol $\Pi^{\mathbb{F}_2}_{\mathsf{OLE-Setup}}$ realizes the OLE seed generation functionality $\mathcal{F}^{\mathbb{F}_2}_{\mathsf{OLE-Setup}}$ with security against semi-honest adversaries in the $(\mathcal{F}_{\mathsf{2PC}}, \mathcal{F}_{\mathsf{DPF}})$-hybrid model.*

Proof. Note that $\Pi^{\mathbb{F}_2}_{\mathsf{OLE-Setup}}$ securely evaluates each step of $\mathsf{PCG}_{\mathsf{OLE.Setup}}$ of $\mathsf{Cons}^{\mathbb{F}_2}_{\mathsf{OLE}}$. The $\mathsf{SPFSS.Gen}$ is implemented via calling the $\mathcal{F}_{\mathsf{DPF}}$ upon each nonzero point. □

Remark D.2. The above protocol considers the ring $\mathcal{R} = \mathbb{F}_{2^k}[\mathtt{X}]/(\mathtt{X}^N - 1)$, and the "additions" are essentially computed over $\mathbb{Z}_N$.

D.4 PCG Setup Protocols from Ring-LPN with Malicious Security

We show a protocol for generating OLE correlations realizing the corruptible OLE functionality $\mathcal{F}^{\mathbb{F}_2}_{\mathsf{mal-OLE}}$ with security against malicious adversaries. The underlying assumption is the Ring-LPN with static leakage, which was introduced in [BCG+20, Section 6.2]. Intuitively, in the distributed setup protocol, an adversary is able to maliciously guess a predicate on the positions of the error vector. Our protocol for OLE over $\mathbb{F}_2$ is given in $\Pi^{\mathbb{F}_2}_{\mathsf{mal-OLE}}$.

Protocol 19: $\Pi^{\mathbb{F}_2}_{\mathsf{mal-OLE}}$

Parameters: Security parameter 1^λ, noise weight $t = t(\lambda)$, length $N = 2^k - 1$, $\mathbb{F} := \mathbb{F}_{2^k} = \mathbb{F}_2(\xi)$ and $\mathcal{R}_k = \mathbb{F}_{2^k}[\mathtt{X}]/(\mathtt{X}^N - 1)$. Let $\mathbb{G} = \mathbb{Z}_N$. Let $\eta \in \mathbb{N}$ such that $\eta \geq \lambda$ and $k \mid \eta$. Let DPF be a distributed point function $\mathsf{DPF} := (\mathsf{DPF.Gen}, \mathsf{DPF.Eval})$ with domain size N and range $\mathbb{F}_{2^k}$ or $\mathbb{F}_{2^\eta}$. We abuse ϕ as a fixed isomorphism map from $\mathcal{R}_k$ to $\mathbb{F}_{2^k}^N$, and from $\mathcal{R}_\eta$ to $\mathbb{F}_{2^\eta}^N$. Assume each variable is shared via a SPDZ-style authenticated sharing, for instance $[\![x]\!] := ([\![x]\!]_0, [\![x]\!]_1)$ and $[\![x]\!]_\sigma := (x_\sigma, M_\sigma[x])$ where $x = x_0 + x_1$, $M_0[x] + M_1[x] = x \cdot (\Delta_0 + \Delta_1)$ and (Δ_0, Δ_1) is a sharing of the global key. For simplicity, the malicious PowerP operation is implicitly represented as $[\![\cdot]\!]^{p^\ell}$. There exists an implicit $\mathsf{ToBinary}$ function call before calling $\mathcal{F}_{\mathsf{mal-DPF}}$. Furthermore, it is given access to the functionalities $\mathcal{F}_{\mathsf{2PC}}$ and $\mathcal{F}_{\mathsf{mal-DPF}}$.
Input: A random element $a \in \mathcal{R}_k$.
Correlation: For $\theta \in [0, k-1]$, output $\mathbf{X}_0^{(\theta)}, \mathbf{Z}_0^{(\theta)}, \mathbf{X}_1^{(\theta)}, \mathbf{Z}_1^{(\theta)} \in \mathbb{F}_2^N$ such that $\mathbf{X}_0^{(\theta)} * \mathbf{X}_1^{(\theta)} = \mathbf{Z}_0^{(\theta)} + \mathbf{Z}_1^{(\theta)}$.
Protocol:

1. P_σ samples random vectors $\mathbf{A}_\sigma^0, \mathbf{A}_\sigma^1 \xleftarrow{\$} \mathbb{G}^t$ and $\mathbf{b}_\sigma^0, \mathbf{b}_\sigma^1 \xleftarrow{\$} (\mathbb{F}^*)^t$. Note that each pair $(\mathbf{A}_\sigma^i, \mathbf{b}_\sigma^i)$ defines a t-sparse element in $\mathcal{R}_k$.
2. Input the position and values. For $\sigma \in \{0,1\}$, $i \in [0,1]$, $j \in [1,t]$,
$[\![\mathbf{A}_\sigma^i[j]]\!] \leftarrow \mathsf{Input}(P_\sigma, \mathbf{A}_\sigma^i[j])$, $[\![\mathbf{b}_\sigma^i[j]]\!] \leftarrow \mathsf{Input}(P_\sigma, \mathbf{b}_\sigma^i[j])$
3. Generate FSS keys for $\left[\!\left[\mathbf{Z}^{(\theta)}\right]\!\right]$ according to Eq. (10). For $i, j \in [1,t]$, $\kappa, \ell \in [0, k-1]$, //Note there exists an implicit $\mathsf{ToBinary}$ function call before calling $\mathcal{F}_{\mathsf{mal-DPF}}$.

$$(K_0^{ij\kappa\ell 0}, K_1^{ij\kappa\ell 0}) \leftarrow \mathcal{F}_{\mathsf{mal-DPF}}(2^\kappa [\![\mathbf{A}_0^0[i]]\!] + 2^\ell [\![\mathbf{A}_1^0[j]]\!], [\![\mathbf{b}_0^0[i]]\!]^{2^\kappa} \cdot [\![\mathbf{b}_1^0[j]]\!]^{2^\ell})$$
$$(K_0^{ij\kappa\ell 1}, K_1^{ij\kappa\ell 1}) \leftarrow \mathcal{F}_{\mathsf{mal-DPF}}(2^\kappa [\![\mathbf{A}_0^0[i]]\!] + 2^\ell [\![\mathbf{A}_1^1[j]]\!], [\![\mathbf{b}_0^0[i]]\!]^{2^\kappa} \cdot [\![\mathbf{b}_1^1[j]]\!]^{2^\ell})$$
$$(K_0^{ij\kappa\ell 2}, K_1^{ij\kappa\ell 2}) \leftarrow \mathcal{F}_{\mathsf{mal-DPF}}(2^\kappa [\![\mathbf{A}_0^1[i]]\!] + 2^\ell [\![\mathbf{A}_1^0[j]]\!], [\![\mathbf{b}_0^1[i]]\!]^{2^\kappa} \cdot [\![\mathbf{b}_1^0[j]]\!]^{2^\ell})$$
$$(K_0^{ij\kappa\ell 3}, K_1^{ij\kappa\ell 3}) \leftarrow \mathcal{F}_{\mathsf{mal-DPF}}(2^\kappa [\![\mathbf{A}_0^1[i]]\!] + 2^\ell [\![\mathbf{A}_1^1[j]]\!], [\![\mathbf{b}_0^1[i]]\!]^{2^\kappa} \cdot [\![\mathbf{b}_1^1[j]]\!]^{2^\ell})$$

4. Generate $\mathbf{X}_\sigma^{(\theta)}$. P_σ computes

$$e_\sigma^0 = \sum_{j \in [1,t]} \mathbf{b}_\sigma^0[j] \cdot \mathtt{X}^{\mathbf{A}_\sigma^0[j]}, e_\sigma^1 = \sum_{j \in [1,t]} \mathbf{b}_\sigma^1[j] \cdot \mathtt{X}^{\mathbf{A}_\sigma^1[j]}$$

and

$$\mathbf{X}_\sigma^{(\theta)} = \phi\Big(\mathsf{Tr}(\xi^{2^\theta} \cdot (a \cdot e_\sigma^0 + e_\sigma^1))\Big).$$

5. Generate $\left[\!\left[\mathbf{Z}^{(\theta)}\right]\!\right]$. Set

$$K_\sigma^{\kappa\ell 0} := \sum_{i,j \in [t]} K_\sigma^{ij\kappa\ell 0}, \quad K_\sigma^{\kappa\ell 1} := \sum_{i,j \in [t]} K_\sigma^{ij\kappa\ell 1},$$
$$K_\sigma^{\kappa\ell 2} := \sum_{i,j \in [t]} K_\sigma^{ij\kappa\ell 2}, \quad K_\sigma^{\kappa\ell 3} := \sum_{i,j \in [t]} K_\sigma^{ij\kappa\ell 3}.$$

For $\theta \in [0, k-1]$, compute

$$\left[\!\left[\mathbf{Z}^{(\theta)}\right]\!\right]_\sigma := \phi\Big(\sum_{\kappa,\ell\in[0,k-1]} \xi^{2^\theta(2^\kappa+2^\ell)} \cdot \big(a^{2^\kappa+2^\ell} \cdot K_\sigma^{\kappa\ell 0} + a^{2^\kappa} \cdot K_\sigma^{\kappa\ell 1} + a^{2^\ell} \cdot K_\sigma^{\kappa\ell 2} + K_\sigma^{\kappa\ell 3}\big)\Big).$$

6. P_σ outputs $\left(\mathbf{X}_\sigma^{(\theta)}, \mathbf{Z}_\sigma^{(\theta)}\right)$. //Note that we only output the sharing of $\mathbf{Z}^{(\theta)}$ without the MAC shares, though we actually get $\left[\!\left[\mathbf{Z}^{(\theta)}\right]\!\right]$.

Theorem D.4. *Assume the hardness of Ring-LPN assumption with static leakage, then the protocol* $\Pi^{\mathbb{F}_2}_{\mathsf{mal-OLE}}$ *implements the functionality* $\mathcal{F}^{\mathbb{F}_2}_{\mathsf{mal-OLE}}$ *in the* $(\mathcal{F}_{\mathsf{2PC}}, \mathcal{F}_{\mathsf{mal-DPF}})$*-hybrid model against malicious adversaries.*

D.5 Authenticated Boolean Triples from Ring-LPN

We define the ideal corruptible functionality for authenticated Boolean triples in $\mathcal{F}^{\mathbb{F}_2}_{\mathsf{Auth-Triple}}$, and give the protocol $\Pi^{\mathbb{F}_2}_{\mathsf{Auth-Triple}}$ that realizes it with malicious security.

Functionality 20: $\mathcal{F}^{\mathbb{F}_2}_{\mathsf{Auth-Triple}}$

Parameters: Security parameter 1^λ, $N \mid 2^k - 1$, $\xi \in \mathbb{F}_{2^k}$ s.t. $\mathbb{F}_{2^k} = \mathbb{F}_2(\xi)$. Let $\eta \in \mathbb{N}$ such that $\eta \geq \lambda$ and $k \mid \eta$.
Functionality:
If both parties are honest,

1. Sample $\Delta_0, \Delta_1 \xleftarrow{\$} \mathbb{F}_{2^\eta}$ and let $\Delta \leftarrow \Delta_0 + \Delta_1$.
2. Sample $\mathbf{x}_0, \mathbf{x}_1, \mathbf{y}_0, \mathbf{y}_1 \xleftarrow{\$} \mathbb{F}_2^{kN}$ and let $\mathbf{x} = \mathbf{x}_0 + \mathbf{x}_1, \mathbf{y} = \mathbf{y}_0 + \mathbf{y}_1$.
3. Let $\mathbf{z} \leftarrow \mathbf{x} * \mathbf{y} \in \mathbb{F}_2^{kN}$.
4. Sample $\mathbf{m}_{x,0}, \mathbf{m}_{y,0}, \mathbf{m}_{z,0} \xleftarrow{\$} \mathbb{F}_{2^\eta}^{kN}$ and let $\mathbf{m}_{x,1} \leftarrow \Delta \cdot \mathbf{x} - \mathbf{m}_{x,0}, \mathbf{m}_{y,1} \leftarrow \Delta \cdot \mathbf{y} - \mathbf{m}_{y,0}, \mathbf{m}_{z,1} \leftarrow \Delta \cdot \mathbf{z} - \mathbf{m}_{z,0}$.
5. For $\sigma \in \{0,1\}$, output $(\Delta_\sigma, \mathbf{x}_\sigma, \mathbf{y}_\sigma, \mathbf{z}_\sigma, \mathbf{m}_{x,\sigma}, \mathbf{m}_{y,\sigma}, \mathbf{m}_{z,\sigma})$ to P_σ.

If P_σ is corrupted,

1. Wait for input $(\Delta_\sigma, \mathbf{x}_\sigma, \mathbf{y}_\sigma, \mathbf{z}_\sigma, \mathbf{m}_{x,\sigma}, \mathbf{m}_{y,\sigma}, \mathbf{m}_{z,\sigma}) \in \mathbb{F}_{2^\eta} \times \mathbb{F}_2^{3kN} \times \mathbb{F}_{2^\eta}^{3kN}$ from the adversary.
2. Sample $\Delta_{1-\sigma} \xleftarrow{\$} \mathbb{F}_{2^\eta}$ and $\mathbf{x}_{1-\sigma}, \mathbf{y}_{1-\sigma} \xleftarrow{\$} \mathbb{F}_2^{kN}$. Set $\Delta \leftarrow \Delta_0 + \Delta_1, \mathbf{x} \leftarrow \mathbf{x}_0 + \mathbf{x}_1, \mathbf{y} \leftarrow \mathbf{y}_0 + \mathbf{y}_1$ and $\mathbf{z} = \mathbf{x} * \mathbf{y}$. Let $\mathbf{m}_{x,1-\sigma} \leftarrow \Delta \cdot \mathbf{x} - \mathbf{m}_{x,\sigma}, \mathbf{m}_{y,1-\sigma} \leftarrow \Delta \cdot \mathbf{y} - \mathbf{m}_{y,\sigma}, \mathbf{m}_{z,1-\sigma} \leftarrow \Delta \cdot \mathbf{z} - \mathbf{m}_{z,\sigma}$.
3. Output $(\Delta_{1-\sigma}, \mathbf{x}_{1-\sigma}, \mathbf{y}_{1-\sigma}, \mathbf{z}_{1-\sigma}, \mathbf{m}_{x,1-\sigma}, \mathbf{m}_{y,1-\sigma}, \mathbf{m}_{z,1-\sigma})$ to the honest party $P_{1-\sigma}$.

Protocol 21: $\Pi^{\mathbb{F}_2}_{\mathsf{Auth-Triple}}$

Parameters: Security parameter 1^λ, noise weight $t = t(\lambda)$, length $N \mid 2^k - 1$, $\mathbb{F} := \mathbb{F}_{2^k} = \mathbb{F}_2(\xi)$ and $\mathcal{R}_k = \mathbb{F}_{2^k}[\mathtt{X}]/(\mathtt{X}^N - 1)$. Let $\mathbb{G} = \mathbb{Z}_N$. Let $\eta \in \mathbb{N}$ such that $\eta \geq \lambda$ and $k \mid \eta$. Let DPF be a distributed point function $\mathsf{DPF} := (\mathsf{DPF.Gen}, \mathsf{DPF.Eval})$ with domain size N and range $\mathbb{F}_{2^k}$ or $\mathbb{F}_{2^\eta}$. We abuse ϕ as a fixed isomorphism map from $\mathcal{R}_k$ to $\mathbb{F}_{2^k}^N$, and from $\mathcal{R}_\eta$ to $\mathbb{F}_{2^\eta}^N$. Assume each variable is shared via a SPDZ-style authenticated sharing, for instance $[\![x]\!] := ([\![x]\!]_0, [\![x]\!]_1)$ and $[\![x]\!]_\sigma := (x_\sigma, M_\sigma[x])$ where $x = x_0 + x_1$, $M_0[x] + M_1[x] = x \cdot (\Delta_0 + \Delta_1)$ and (Δ_0, Δ_1) is a sharing of the global key. For simplicity, the malicious PowerP operation is implicitly represented as $[\![\cdot]\!]^{2^\ell}$. There exists an implicit $\mathsf{ToBinary}$ function call before calling $\mathcal{F}_{\mathsf{mal-DPF}}$. Furthermore, it is given access to the functionalities $\mathcal{F}_{\mathsf{2PC}}$ and $\mathcal{F}_{\mathsf{mal-DPF}}$.
Input: A random element $a \in \mathcal{R}_k$.
Protocol:

1. The parties sample the error vector and input them. For $i \in \{0,1\}$, P_σ samples $\mathbf{A}_\sigma^{x,i}, \mathbf{A}_\sigma^{y,i} \xleftarrow{\$} \mathbb{G}^t$ and $\mathbf{b}_\sigma^{x,i}, \mathbf{b}_\sigma^{y,i} \xleftarrow{\$} (\mathbb{F}^*)^t$. For $i \in \{0,1\}$,$j \in [1,t]$,
   (i) $[\![\mathbf{A}_\sigma^{x,i}[j]]\!] \leftarrow \mathsf{Input}(\sigma, \mathbf{A}_\sigma^{x,i}[j])$, $[\![\mathbf{b}_\sigma^{x,i}[j]]\!] \leftarrow \mathsf{Input}(\sigma, \mathbf{b}_\sigma^{x,i}[j])$.
   (ii) $[\![\mathbf{A}_\sigma^{y,i}[j]]\!] \leftarrow \mathsf{Input}(\sigma, \mathbf{A}_\sigma^{y,i}[j])$, $[\![\mathbf{b}_\sigma^{y,i}[j]]\!] \leftarrow \mathsf{Input}(\sigma, \mathbf{b}_\sigma^{y,i}[j])$.
   Then P_σ obtains $\left\{ [\![\mathbf{A}^{x,i}[j]]\!]_\sigma, [\![\mathbf{b}^{x,i}[j]]\!]_\sigma, [\![\mathbf{A}^{y,i}[j]]\!]_\sigma, [\![\mathbf{b}^{y,i}[j]]\!]_\sigma \right\}_{i\in\{0,1\}, j\in[1,2t]}$.
2. Generate FSS keys for $\left(\left[\!\left[\mathbf{X}^{(\theta)} \right]\!\right], \left[\!\left[\mathbf{Y}^{(\theta)} \right]\!\right] \right)$ according to Eq.(9). For $i \in \{0,1\}$, $j \in [1, 2t]$, $\ell \in [0, k-1]$, //Note there exists an implicit $\mathsf{ToBinary}$ function call before calling $\mathcal{F}_{\mathsf{mal-DPF}}$.

$$(K_0^{x,i,j,\ell}, K_1^{x,i,j,\ell}) \leftarrow \mathcal{F}_{\mathsf{mal-DPF}}(2^\ell \cdot \left[\!\left[\mathbf{A}^{x,i}[j] \right]\!\right], \left[\!\left[\mathbf{b}^{x,i}[j] \right]\!\right]^{2^\ell})$$

$$(K_0^{y,i,j,\ell}, K_1^{y,i,j,\ell}) \leftarrow \mathcal{F}_{\mathsf{mal-DPF}}(2^\ell \cdot \left[\!\left[\mathbf{A}^{y,i}[j] \right]\!\right], \left[\!\left[\mathbf{b}^{y,i}[j] \right]\!\right]^{2^\ell})$$

3. Generate FSS keys for $\left[\!\left[\mathbf{Z}^{(\theta)} \right]\!\right]$ according to Eq. (10). For $i, j \in [1, 2t]$, $\kappa, \ell \in [0, k-1]$, //Note there exists an implicit $\mathsf{ToBinary}$ function call before calling $\mathcal{F}_{\mathsf{mal-DPF}}$.

$$(K_0^{ij\kappa\ell 0}, K_1^{ij\kappa\ell 0}) \leftarrow \mathcal{F}_{\mathsf{mal-DPF}}(2^\kappa \left[\!\left[\mathbf{A}^{x,0}[i] \right]\!\right] + 2^\ell \left[\!\left[\mathbf{A}^{y,0}[j] \right]\!\right], \left[\!\left[\mathbf{b}^{x,0}[i] \right]\!\right]^{2^\kappa} \cdot \left[\!\left[\mathbf{b}^{y,0}[j] \right]\!\right]^{2^\ell})$$

$$(K_0^{ij\kappa\ell 1}, K_1^{ij\kappa\ell 1}) \leftarrow \mathcal{F}_{\mathsf{mal-DPF}}(2^\kappa \left[\!\left[\mathbf{A}^{x,0}[i] \right]\!\right] + 2^\ell \left[\!\left[\mathbf{A}^{y,1}[j] \right]\!\right], \left[\!\left[\mathbf{b}^{x,0}[i] \right]\!\right]^{2^\kappa} \cdot \left[\!\left[\mathbf{b}^{y,1}[j] \right]\!\right]^{2^\ell})$$

$$(K_0^{ij\kappa\ell 2}, K_1^{ij\kappa\ell 2}) \leftarrow \mathcal{F}_{\mathsf{mal-DPF}}(2^\kappa \left[\!\left[\mathbf{A}^{x,1}[i] \right]\!\right] + 2^\ell \left[\!\left[\mathbf{A}^{y,0}[j] \right]\!\right], \left[\!\left[\mathbf{b}^{x,1}[i] \right]\!\right]^{2^\kappa} \cdot \left[\!\left[\mathbf{b}^{y,0}[j] \right]\!\right]^{2^\ell})$$

$$(K_0^{ij\kappa\ell 3}, K_1^{ij\kappa\ell 3}) \leftarrow \mathcal{F}_{\mathsf{mal-DPF}}(2^\kappa \left[\!\left[\mathbf{A}^{x,1}[i] \right]\!\right] + 2^\ell \left[\!\left[\mathbf{A}^{y,1}[j] \right]\!\right], \left[\!\left[\mathbf{b}^{x,1}[i] \right]\!\right]^{2^\kappa} \cdot \left[\!\left[\mathbf{b}^{y,1}[j] \right]\!\right]^{2^\ell})$$

4. Generate $(\left[\!\left[\mathbf{X}^{(\theta)} \right]\!\right], \left[\!\left[\mathbf{Y}^{(\theta)} \right]\!\right])$. Set

$$K_\sigma^{x,i,\ell} := \sum_{j\in[1,2t]} K_\sigma^{x,i,j,\ell}, \quad K_\sigma^{y,i,\ell} := \sum_{j\in[1,2t]} K_\sigma^{y,i,j,\ell}.$$

For $\theta \in [0, k-1]$, compute

$$\left[\!\left[\mathbf{X}^{(\theta)}\right]\!\right]_\sigma := \phi\Big(\sum_{\ell\in[0,k-1]} \xi^{2^{\theta+\ell}} \cdot (a^{2^\ell} \cdot K_\sigma^{x,0,\ell} + K_\sigma^{x,1,\ell})\Big)$$

$$\left[\!\left[\mathbf{Y}^{(\theta)}\right]\!\right]_\sigma := \phi\Big(\sum_{\ell\in[0,k-1]} \xi^{2^{\theta+\ell}} \cdot (a^{2^\ell} \cdot K_\sigma^{y,0,\ell} + K_\sigma^{y,1,\ell})\Big)$$

5. Generate $\left[\!\left[\mathbf{Z}^{(\theta)}\right]\!\right]$. Set

$$K_\sigma^{\kappa\ell 0} := \sum_{i,j\in[2t]} K_\sigma^{ij\kappa\ell 0}, \quad K_\sigma^{\kappa\ell 1} := \sum_{i,j\in[2t]} K_\sigma^{ij\kappa\ell 1},$$

$$K_\sigma^{\kappa\ell 2} := \sum_{i,j\in[2t]} K_\sigma^{ij\kappa\ell 2}, \quad K_\sigma^{\kappa\ell 3} := \sum_{i,j\in[2t]} K_\sigma^{ij\kappa\ell 3}.$$

For $\theta \in [0, k-1]$, compute

$$\left[\!\left[\mathbf{Z}^{(\theta)}\right]\!\right]_\sigma := \phi\Big(\sum_{\kappa,\ell\in[0,k-1]} \xi^{2^\theta(2^\kappa+2^\ell)} \cdot \big(a^{2^\kappa+2^\ell} \cdot K_\sigma^{\kappa\ell 0} + a^{2^\kappa} \cdot K_\sigma^{\kappa\ell 1} + a^{2^\ell} \cdot K_\sigma^{\kappa\ell 2} + K_\sigma^{\kappa\ell 3}\big)\Big).$$

6. P_σ outputs $\Big(\left[\!\left[\mathbf{X}^{(\theta)}\right]\!\right]_\sigma, \left[\!\left[\mathbf{Y}^{(\theta)}\right]\!\right]_\sigma, \left[\!\left[\mathbf{Z}^{(\theta)}\right]\!\right]_\sigma\Big)$.

Theorem D.5. *Assume the hardness of Ring-LPN assumption with static leakage, then the protocol* $\Pi^{\mathbb{F}_2}_{\mathsf{Auth-Triple}}$ *implements the functionality* $\mathcal{F}^{\mathbb{F}_2}_{\mathsf{Auth-Triple}}$ *in the* $(\mathcal{F}_{\mathsf{2PC}}, \mathcal{F}_{\mathsf{mal-DPF}})$*-hybrid model against malicious adversaries.*